CONTENTS

	Page	
SUMMARY	1	1/A6
INTRODUCTION	1	1/A6
GENERAL CONSIDERATIONS	2	1/A7
DEBUG PROGRAM CAPABILITIES	4	1/A9
Initialization	4	1/A9
Addressing	4	1/A9
Information Display	5	1/A10
Program Modifications	6	1/A11
Execution and Analysis	6	1/A11
PROGRAM DESCRIPTION	8	1/A13
Command Structure	8	1/A13
Name Assignments	8	1/A13
Address Formats	9	1/A14
Debug Assembler	10	1/B1
Scaling	11	1/B2
DEBUG COMMANDS	12	1/B3
Initialization Command - YDEBUG	13	1/B4
Source-Name Input Command - UDEBUG	13	1/B4
Sector Base Statistics Command - BDEBUG	14	1/B5
Display Command - PDEBUG	15	1/B6
Modification Command - CDEBUG	16	1/B7
Program Addition Command - ADEBUG	17	1/B8
Subroutine Specification and Execution Command - VDEBUG	17	1/B8
Program Execution Command - XDEBUG	17	1/B8
Breakpoint Command - SDEBUG	19	1/B10
Return Command - RDEBUG	19	1/B10
Transfer Command - TDEBUG	20	1/B11
CONCLUSIONS	21	1/B12
APPENDIX - DESCRIPTION OF DEBUG PROGRAM	22	1/B13
REFERENCES	26	1/C3

UN 5 1979

from 830-11-15

NAS 1.60:1441

COMPLETED

NASA Technical Paper 1441

Interactive Debug Program for Evaluation and Modification of Assembly-Language Software

Dale J. Arpasi

APRIL 1979

NASA

Microfilmed From
Best Available Copy

73

NASA Technical Paper 1441

Interactive Debug Program for Evaluation and Modification of Assembly-Language Software

Dale J. Arpasi
Lewis Research Center
Cleveland, Ohio

National Aeronautics
and Space Administration

**Scientific and Technical
Information Office**

1979

Blank Page

CONTENTS

	Page
SUMMARY	1
INTRODUCTION	1
GENERAL CONSIDERATIONS	2
DEBUG PROGRAM CAPABILITIES	4
Initialization	4
Addressing	4
Information Display	5
Program Modifications	6
Execution and Analysis	6
PROGRAM DESCRIPTION	8
Command Structure	8
Name Assignments	8
Address Formats	9
Debug Assembler	10
Scaling	11
DEBUG COMMANDS	12
Initialization Command - YDEBUG	13
Source-Name Input Command - UDEBUG	13
Sector Base Statistics Command - BDEBUG	14
Display Command - PDEBUG	15
Modification Command - CDEBUG	16
Program Addition Command - ADEBUG	17
Subroutine Specification and Execution Command - VDEBUG	17
Program Execution Command - XDEBUG	17
Breakpoint Command - SDEBUG	19
Return Command - RDEBUG	19
Transfer Command - TDEBUG	20
CONCLUSIONS	21
APPENDIX - DESCRIPTION OF DEBUG PROGRAM	22
REFERENCES	26

SUMMARY

Using the digital computer in real-time aircraft propulsion control systems requires extensive use of assembly-language programming. Because this language is far removed from the actual control equations, significant control-software evaluation and debugging problems often occur. Therefore, software is needed to alleviate these problems.

Presently available debug software is not sufficient to raise operator-computer interaction to the level where software evaluation and debugging become acceptable. Therefore, a higher level debug program was developed. This program uses information supplied by the computer's assembler and loader and by the operator to simplify problem recognition and to permit straightforward modifications to the software being debugged.

Addressing techniques are used that allow operator-computer interaction by means of easily recognizable names and relative addresses as they appear in the source listing. Many data formats are available, including one that specifies data in engineering units. An instruction coder and decoder allow program instructions to be specified by their assembler mnemonic representation and to be evaluated during execution of the program that is to be debugged. Thus, the operator need not be concerned with actual machine coding. Execution terminators are included that, when used, will stop debug execution of the program if certain prespecified conditions are encountered. The operator may specify execution termination when conditions are encountered that modify the program counter or the selected memory locations or that cause arithmetic overflows. Other features incorporated into the debug program include an on-line assembler that makes it easier to modify the program that is to be debugged.

The debug program is described in detail in this report. The debug commands, their sequencing, and their options are described and illustrated. Functional diagrams of the debug program are given in the appendix.

INTRODUCTION

Recent advances in digital computer hardware have enhanced its use in control applications. Controls requiring extensive computations and logical decisions, such as those for modern aircraft propulsion systems, are particularly well suited for implementation on a digital computer. Aircraft propulsion controls, however, are usually bound by stringent computation-time limits.

Computation speed depends on the programming language used to implement the control equations. In general, the more removed the programming language is from the basic machine instruction set, the slower is its computation. Therefore, computation-time limits often preclude the use of higher order languages that offer a format close to the actual control equations and force the programmer to use assembly language. Because assembly language is far removed from the actual control equations. significant program debugging problems may be encountered.

Debugging consists of analyzing and modifying the assembly-language program. It may be done through direct operator-computer interaction, which is itself very prone to error because of the bookkeeping involved, or through a buffer program that raises this interaction to a more comprehensible level. Although such programs (referred to as debug programs) are usually available with every computer, they are generally not extensive enough to meet the needs of the control-system programmer. The debug program described in this report offers more versatile addressing, display, execution, and on-line assembly capabilities than previous debug programs. It was written for the Honeywell HDC-601 flight-qualified computer and the Honeywell DDP-516/316 computer. The philosophy and techniques used in developing the program are described in detail so that the concepts can be applied to other computers. Because this report is also intended as a users guide, operational details and examples are given.

The control-software debugging process requires operational data for software evaluation. These data are generally collected by running the software in conjunction with the system to be controlled (or a real-time simulation of the system). The data-collection process is made simpler by a data-collecting and display program called INFORM (ref. 1). This program is a high-level, operator-computer interaction program described in this report offers more versatile addressing, display, execution, and on-line assembly capabilities than previous debug programs. It was written for the software evaluation and modification capabilities to the control programmer.

GENERAL CONSIDERATIONS

The debug program was written for the Honeywell HDC-601 flight-qualified computer and the Honeywell DDP-516/316 computer. The program depends on the computer architecture and instruction set. The computers are described in detail in references 2 and 3. The HDC-601 uses a 16-bit word with the most significant bit designated as bit 1. The computer has four manipulative registers:

(1) A register (AREG) - the primary arithmetic and utility register

(2) B register (BREG) - the secondary arithmetic register

(3) X register (XREG) - the index rigister

(4) KEYS - the register used to contain machine status information

The computer uses sectorized addressing: Memory is divided into 512-word sectors. Communication within a sector is done directly, but communication between sectors must be done indirectly through intersector references. The one exception is that communication with the first memory sector can be made directly from any sector. Intersector references, therefore, can be stored in the first sector. The computer's assembler and loader also allow an area within each sector to be set aside for intersector references. This area is referred to as the sector base area.

The HDC-601 instruction set can be divided into two categories: those requiring an operand, and those not requiring an operand. Instructions requiring an operand are

(1) Memory reference instructions

(2) Input-output instructions

(3) Shift instructions

Each instruction has a three-character, assembly-language mnemonic representation and a machine-language operational code (op-code). These considerations and those of the preceding paragraph are important to the structuring of the debug program and must be considered in using the program concepts to formulate debug programs for other digital computers.

The basic philosophy of the debug program development was to use assembler-, loader-, and operator-supplied information to provide operator-computer interaction in an understandable format. The assembler supplies a source listing and a table of source names. The debug program uses these names to simplify operator address recognition. The loader supplies the relocation base addresses and the locations of sector base areas. The debug program uses them to eliminate the need for absolute address specification and to efficiently automate program modifications. Through INFORM, the operator may assign scale factors and names to memory addresses. This allows data to be specified directly in engineering units.

The computer's instruction set is incorporated into the debug program. Because both the mnemonics and the op-codes are included, program modifications can be structured in assembly language without resorting to machine coding. Conversely, machine coding for program listings can be automatically interpreted in an easily understandable format.

Because the command structure of the debug program is fully compatible with the command structure of INFORM (ref. 1), the programmer can incorporate any INFORM commands with any debug commands for an evaluation and modification program that suits his needs.

DEBUG PROGRAM CAPABILITIES

The capabilities of the debug program are

(1) Initialization
(2) Addressing
(3) Information display
(4) Program modifications
(5) Execution and analysis

Each capability is discussed in general in this section to give an understanding of the program concepts. Detailed program descriptions and operational examples are given in subsequent sections.

Initialization

The debug program must be initialized before execution to provide the information necessary to its operation. That is, memory areas must be specified to contain

(1) The debug name tables (source names assigned to addresses within the program to be debugged)
(2) The debug on-line assembler's buffer (the area used to assemble operator-supplied instructions)
(3) Program additions (the area assigned for additions to the program being debugged)
(4) The INFORM name tables (names and scale factors assigned to any memory address when INFORM commands are being used)

In addition, the operator may specify a protected memory area that cannot be violated during debug execution of his software. After memory-area assignments are specified, the operator may specify the sector base areas and fill the name tables. The debug name tables may be filled manually (e.g., keyboard) or automatically (e.g., paper tape) by entering the table of source names obtained during assembly of the program that is to be debugged. Any number of programs or subroutines can be debugged simultaneously, and the source-name assignment is limited only by the area set aside.

Addressing

Flexible operator-address specification is offered in the debug program. It allows referencing of an absolute octal address, a source name, an INFORM name, and a relocatable octal address. Arithmetic-address stringing is permitted in combinations of the preceding. In addition, special addressing - including undefined addressing, index-

ing, and indirect address specification - is available for instruction operands during on-line assembly.

Two locations, called the first and last address counters, are used to specify the initial and final address limits for debug commands. The first address counter is also used to establish the base address for address displacements. When a source name or a relocatable octal address is specified, the first address counter is used to determine which program is being debugged and which source-name table is referenced. The relocation base for this program (specified during initialization) is added to the specified address to form the absolute address. With this address mode the operator may easily work from the source listing of the program currently being debugged. An address format is available to reference names external to the current program.

Information Display

An information display package is contained in the debug program. Any of the four registers (AREG, BREG, XREG, and KEYS) or any memory location or series of memory locations can be displayed in an operator-selected format. Display formats are available for octal integers, decimal single- and double-precision integers, and decimal floating-point numbers. (Because all decimal displays can be descaled, the display can be in engineering units.) Display formats are also available that specify the address as containing ASCII characters or a direct-address constant.

Additional display formats are available for memory display only. Display of the effective address of a direct-address constant (deciphering of the indirect chain) and display of block-zero storage (the number of successive addresses containing zero) may be specified. In addition, the address may be specified to contain an instruction that will cause the display of the mnemonic and operand, if any. If the instruction is a memory-referencing instruction, the status of the index and indirect indicators is also displayed.

Two display modes are available: the print mode and the list mode. The print mode simply lists the address and its contents in the specified format. The list mode produces a listing similar to that of the machine assembler. Only the print mode is available for register display.

It is often preferable to separate the display commands from the information to be displayed. This capability is incorporated in the display package: The operator can input commands from one unit (e.g., teletype) and have the display appear on another unit (e.g., line printer).

Program Modifications

The debugging process often involves significant program and data modifications. Without a comprehensive debug package the operator is forced either to frequently assemble the source program and reload the resulting object program or to modify the machine language. Both alternatives are time consuming and the latter is particularly prone to error. By incorporating an on-line assembler, the debug program discussed in this report provides a quick turnaround for all program modifications.

The debug assembler is similar to the machine assembler in that location, instruction, and address fields must be specified. The location field is used to specify the relative location of the entry in the program modification. The instruction field is used to specify the instruction mnemonic and for indirect referencing. The address field is used to specify the instruction operand if necessary.

All the machine-instruction mnemonics are available. In addition, pseudo-operation mnemonics (pseudo-ops) are available to specify octal entries, scaled or unscaled decimal integer or floating-point entries, direct-address constants, ASCII information, and block storage requirements.

Address specification has been defined. Special addressing is available to reference location-field entries or to postpone address definition. All addresses must be defined before the program modification input is completed.

When the operator signals the input to be complete, the program modifications are assembled. The machine coding is formed and the required intersector referencing is computed from the sector base information supplied during initialization. A source listing is then provided for operator verification. The operator may revise the modifications or signal their incorporation into this program.

If the modifications are actually additions to the program, they are placed in the program addition area specified during initialization. Linkages between the programs and the addition are automatically determined.

At any time the operator may transfer the modified program from memory to permanent storage (e.g., paper tape). Verification and loading capabilities are incorporated. Transfer from one memory location to another, with masking of selected bits, is permitted.

Execution and Analysis

The debug program provides a method of execution and analysis that minimizes the possibility of damage to the program to be debugged and maximizes the execution options. The program to be debugged is not executed directly in the debug execution routine. Rather, each instruction is lifted from the program, analyzed, and rebuilt for

execution inside the routine. All machine registers are synthesized. The first address counter is used as the pseudo program counter, and internal locations are set aside as the pseudo instruction register. The registers AREG, BREG, XREG, and KEYS are also synthesized as pseudoregisters. With this method of execution, control never leaves the debug execution routine. By using instruction analysis combined with the execution mode commands, the operator can avoid inadvertent destruction of his program by programming errors.

When the content of the first address counter is set, the execution routine returns the vital statistics of the instruction to be executed: a complete listing of the instruction, the contents of the direct address of memory reference instructions, the effective address of these instructions (deciphering both indirect chains and indexing), and the contents of the pseudoregisters. The operator is advised if the instruction is not executable or if a protected area is about to be violated.

After the listing, the operator can transfer control to another debug routine to obtain additional information, to modify his program, or to change the pseudoregisters. Alternatively, he can execute his program either on an instruction-to-instruction basis or by using one of the following execution modes:

(1) Execute to the termination address specified by the last address counter
(2) Execute to a memory-modifying instruction
(3) Execute to a program-counter-modifying instruction
(4) Execute until an overflow condition exists
(5) Any combination of the above

A permissive memory area may be specified to change the execution termination by memory-modifier instructions. Thus, the operator can prohibit data storage outside a selected program area. When a terminator is encountered, the described listing is produced and the options are again available.

Although breakpoints are not necessary for execution, they are necessary to terminate actual execution of the operator's program. These breakpoints are used to transfer program control from the operator's program to the debug program. Debug commands are available to insert and delete breakpoints and to transfer control back to the operator's program. Control can be transferred from the debug program to the operator's program either directly or through a preset initial-condition program.

A debug routine is used to build on-line and execute subroutines within the debug program. These subroutines can be used to supply initial conditions for control transfers or to structure additional debug routines as desired by the operator.

PROGRAM DESCRIPTION

This section provides background information for understanding debug command operation. The debug commands are described in the next section. The various debug routines are described in detail in the appendix.

Command Structure

The debug program begins with the command structure. Command characters are deciphered here, and the appropriate command routine is entered from this structure. Most returns, on completion of command execution, and most error returns are made to the command struture. The logic is shown in figure 1. The command structure allows the operator to specify INFORM commands as well as debug commands. The INFORM commands are described in reference 1 and are designated by using nonalphanumeric keyboard characters. Debug commands are specified by using control-alphanumeric characters. Each command character is assigned a subroutine name. When the operator specifies a command character, the corresponding command subroutine is determined and program control is transferred to this subroutine.

Since all command characters are assigned subroutine names, unwanted commands must be satisfied with returns to the command structure when loading the program. The operator can therefore tailor an INFORM-debug program to suit a particular need during the program-loading process.

As indicated in figure 1, command operands can be prespecified in terms of a name. Prespecified operands are used for INFORM commands only. This feature is useful for debug commands, however, in that the INFORM name table may be referenced by the debug address specification routines. The INFORM names differ from debug names in that scale factors as well as addresses may be assigned to the names. These statistics are assigned in the command structure. Debug name tables are filled during execution of a particular command.

Name Assignments

The operator can assign names to any memory location. A name can have as many as five alphanumeric characters. Each character is packed in truncated ASCII (six binary bits per character) into two memory words. Figure 2 illustrates the method used. The most significant $2\frac{1}{3}$ characters are packed into the primary name word, and the least significant $2\frac{2}{3}$ characters are packed into the secondary name word. Parallel tables are used to contain the primary and secondary name words. A third parallel table is used to hold the address assignment of the name.

Figure 3 illustrates the structure of the name tables for debug names. Because multiple sequential tables are allowed for debug names, the operator can assign a list of names that correspond to the source listing of each program to be debugged. The first location of each primary-name-word table is set to zero to indicate the start of a new name list. The first location of each secondary-name-word table is used to specify the number of names in the name list. The first location of each address table contains the relocation base of the program, which is determined during program loading.

Address Formats

Addressing is used to specify operands for the debug commands and to satisfy the address requirements of the debug assembler. Several format options are available to the operator through address delimiters and postaddress delimiters.

An address delimiter may be specified instead of a name. Allowable address delimiters are given in table I. Delimiters "APOSTROPHE" and "/" must be directly followed by an octal address. Delimiters "*" and "SPACE" allow the first address counter to be referenced as an address. Delimiters "$" and ":" may only be used for assembler addresses (not for command operands) and must be followed by a decimal statement number. With the exception of the "SPACE" delimiter, all address delimiters require entry of a postaddress delimiter.

After the initial address is determined, the postaddress delimiter is interrogated. These delimiters are given in table II. Delimiters "+" and "-" of table II are used for address stringing. An address string consists of an initial address entry followed by any number of octal or decimal numbers or the ASCII "B" character separated by the "+" or "-" delimiters. Specifying B in the address string causes an external bias address (set through an INFORM command) to be inserted in the string.

The "," delimiter is used to specify assembler address indexing and is prohibited for command operands. This delimiter must be directly followed by the "SPACE" postaddress delimiter.

Failure to enter a proper address or postaddress delimiter will cause an error message, followed by an error return, to be issued. Failure to specify a name that exists in the name table specified in the address initialization routines will also result in an error message and an error return. One exception to this occurs if the name is terminated with the ASCII "." delimiter. In this case, the name is assumed to be unique, and all debug name tables are searched. If the name is not located, the INFORM name table is searched. If it is still not located, an error message and error return are issued. If the name is found, the name table containing that name is determined and control returns to the operator entry point. The operator can then specify a name contained in this new name table. In this way, access is allowed to any name in any name table.

Debug Assembler

The debug assembler is used in all debug commands requiring on-line instruction coding. The sequenced operation requires source-statement input in a format like that used in the regular machine assembler (location, instruction, and address fields). When assembly is completed, a source listing is provided for operator verification.

The location field is five spaces long. It is used to structure the statement entries according to decimal statement numbers as well as to define previously undefined addresses. Location-field entries are also used to signal completion of source-statement entry, to delete statements, and to abort statement entry by a return to the debug command structure. Allowable location-field entries are given in table III. Decimal integers n are used to specify the location of a source statement within the source-statement structure. Thus the operator can easily modify previous statement entries. All statement entries must be sequentially numbered, starting with zero.

Previously undefined address-field entries must be defined before completion of source-statement entry is signaled. They are defined by referencing the undefined address $n in the location field. The location-field entry number immediately following this reference replaces the undefined address as it occurs in all previous instruction entries.

The maximum source-statement number (limited by the size of the assembler buffer to 64) is saved in an instruction counter and used to specify the number of memory locations used by the operator's program. This number may be modified by the "M" command in the location field. The "\" command is used to specify the completion of a source-statement entry, and the "R" command is used to abort the source-statement entry.

After a source-statement number is entered in the location field, the program spaces the input device to the instruction field. The instruction field is six spaces long. Allowable instruction-field entries are given in table IV. Any machine instruction mnemonic or any posudo-op mnemonic given in table V is allowed. Each ASCII character of a mnemonic entry is truncated to five binary bits. The alphabet is thus translated into binary numbers ranging from 00001 for A to 11010 for Z. The ASCII number characters from 1 to 4 are translated by subtracting octal 226 from the ASCII code. These numbers are then binary coded as 11011 to 11110. Since each mnemonic is three characters long, it can be represented by a single 16-bit word (see ref. 2 for HDC-601 instruction mnemonics).

Indirect addressing is specified by appending the ASCII "*" character to the instruction mnemonic. This character signals the end of the instruction-field entry. If indirect addressing is not desired, the ASCII "SPACE" character is used to terminate

the mnemonic and to signal the end of the instruction-field entry. The input device is then spaced to the address field if an instruction operand is required.

Assembler addresses are available to reference as yet undefined addresses or source-statement numbers. These are "$n" and ":n," respectively, where n is a decimal integer as defined for the location field.

The pseudo-operations of table V allow the operator to specify data formats and direct-address constants and to reserve block-storage locations. Each pseudo-op requires an address-field entry. Octal integer data can be entered in unsigned 6-digit format (negative numbers are represented in two's complement). Decimal data can be scaled by terminating the entry with a ASCII "/" character followed by a scale factor (see the section Scaling). An ASCII data entry must be preceded by a decimal number that indicates the number of locations it will consume (two characters per location). Any locations not filled by the operator will be filled with "SPACE" characters. Indirect addressing and indexing are allowed. For block storage, the operator specifies the decimal number of locations to be reserved.

When the operator signals completion of source-statement entry "\", the source program is assembled. The source program, as well as all intersector reference requirements, is listed for operator verification. The listing format for memory reference instructions is given in figure 4. The listing format for other types of instructions and pseudo-operations is similar, with the exception of the address field. If the operator is satisfied, the resulting machine-coded object program is transferred from the assembler buffer to permanent storage. If the listing is unsatisfactory, the operator can either edit the source program or cancel the job.

Scaling

The operator can designate scale factors to be assigned to input or output data. This feature is available for data display or in conjunction with any of the assembler's decimal pseudo-operations (DEC, FPC, and DPC). Scale factors can be specified by a number of means, as given in table VI: Ratios giving the number of engineering units per machine unit may be specified directly. A ratio may also be specified to be that entered for an INFORM name. A scale factor, representing the ratio of voltage to machine units of the analog-to-digital converter in the control system, is stored internally in the program and can be referenced for any variable. Binary scaling is permitted. The ratio of engineering to machine units for binary scaling is $2^{nn}/32\,768$, where nn is a decimal number specified by the operator. It is also possible to default the scale factor to the last entered value.

DEBUG COMMANDS

This section describes the command routines and their use. Eleven commands are available in the debug program. The command structure can be modified to add or delete commands according to need. Command routines are entered by using ASCII control characters (represented by underscoring the corresponding alphabet character in all tables and figures). Table VII lists the debug command routines, their entry characters, and their descriptions and references their functional diagrams (given in figs. 17 to 40).

Completion of each command routine (except RDEBUG, to be described) returns program control to the command structure (fig. 17). This is indicated by the ASCII "←" character. In many cases (particularly in specifying command operands), operator error also causes a return to the command structure. Other errors may simply cause a return to the start of the current entry. The operator can intentionally cause an error return to the command structure at any time by using the ASCII "#" character. One other special error character is used in the program: the ASCII "RUB OUT" character. This character is used in name specification to cancel previously input name characters and to start again. Debug error messages, their descriptions, and their consequences are summarized in table VIII.

Figure 5 presents an assembly-language program to be debugged and illustrates the use of each debug command. It is not intended for any purpose other than illustration. The figure gives the location in memory of each instruction or data word, the octal contents of the location, the name assigned to the location if any, the instruction or pseudo-op mnemonic as defined in reference 2, and the address-field reference. The purpose of each instruction is stated. The program consists of a main program starting at X1 and a subroutine starting at Y1. The X1 program has a sector base area starting at `14735. In this area, one intersector reference location is used and two locations are available for use. The Y1 subroutine is located in a different sector (`15000) than the X1 program and has no sector base area. The subroutine name YYY and its starting location Y1 are equivalent and refer to the same location.

Each command routine is described and illustrated in this section. The sequencing steps and options for each command routine are also given in terms of required operator entries and terminations. The program's response, if any, to each entry is indicated. Certain sequencing steps are common to many of the command routines. These steps are used for address specification, scale-factor specification, and on-line assembly. Sequencing for these operations is given in tables IX, X, and XI, respectively.

Initialization Command - YDEBUG

With the command routine YDEBUG the operator can reserve memory locations to contain

(1) INFORM name and statistics tables

(2) Debug name and address tables

(3) On-line program additions

(4) Debug assembler buffer

In addition, the operator can protect a memory area from violation during program execution when using XDEBUG. This memory area usually is that containing the debug program but can be any selected memory area.

The sequencing and use of this command are given in figure 6. On entry the operator must specify the desired area of initialization (INFORM, debug, or protect). Separate entry must be made for each initialization area.

INFORM initialization requires that the start of its tables (e.g., `16000) and the number of names allowed (e.g., `24) be specified. Since tables must be reserved for the primary and secondary name words and the address and scale-factor (two words) assignments, the actual number of machine locations reserved for INFORM name specification is five times the number of names to be entered plus overhead. After specification, the next available memory location (e.g., `16426) is displayed for the operator.

Debug initialization requires that the program addition area (e.g., `16426) be specified. This area must be contained within a single sector. The assembler buffer must immediately follow the program addition area (e.g., `16700). Therefore, specification of this buffer's location limits the size of the program addition area. The assembler buffer is prespecified to be 64 locations long. Debug name table initialization requires that both the first location (e.g., `17000) and the number of names to be entered (e.g., `24) be specified. The actual number of locations reserved for debug names is three times the number of names to be entered. After specification, the next available memory location (e.g., `17074) is displayed for the operator.

Protect initialization requires that the initial and final locations of the area to be protected (e.g., `15000 ← `15003) be specified. If these locations are not initialized, no area will be protected.

Source-Name Input Command - UDEBUG

With the command routine UDEBUG the operator can specify debug source-name and address assignments. The sequencing and use of this command are given in figure 7. Names assigned to different programs or subroutines must be specified on separate entries into UDEBUG. On entry the operator must specify the relocation base of

the program (e.g., `14740 for program XXX and `15000 for program YYY). Name and address assignments can then be made. In general, names should correspond to the names in the operator's source program, with at least one name being unique to this program (e.g., the program name XXX). This allows a specific name table to be referenced when more than one program is being debugged. More than one name can be assigned to a location (e.g., XXX and XI assigned to `14740). All name entries are terminated with a space.

The address entry must be made in octal and preceded by a zero. Any characters, except "W," between the name terminator and the start of the address will be ignored. Relocatable addresses are terminated by a "SPACE" or a "CARRIAGE RETURN." These are stored directly in the address table (e.g., `00 represents location `14740). Absolute addresses must be terminated by "A." The relocation base is subtracted from the entry before storage (e.g., `014775A is stored as `35).

Completion of the name and address entries is signaled by a "W" in the address field. Any name may precede this entry. This termination corresponds to the source-name-table termination of the HDC-601 assembler listing "0000 WARNING OR ERROR FLAGS"). When the name and address assignments are completed, the accumulated number of locations used in the primary-name-table area is displayed in octal. This includes the initial location of each table used for table statistics.

At any time before entering a name, the operator may change the input device number by starting a line with ";" character. This was not done in the example since all names were entered from a single device. The command routine UDEBUG will ignore any line beginning with "*" and ending with a "CARRIAGE RETURN." This allows for the insertion of comments.

Sector Base Statistics Command - BDEBUG

With the command routine BDEBUG the operator can initialize the sector base tables used by the debug assembler to form intersector references. The sequencing and use of this command are given in figure 8. For each sector used by the programs to be debugged, the operator should describe any area used for intersector referencing by the loader. This allows the assembler to use existing references and thereby optimize memory usage. If additional locations adjacent to this area are available for new references, this should also be indicated. As a minimum, an area in sector zero should be set aside for the creation of intersector references. This sector is used as the default sector if no other area is available.

Ninety-six locations are reserved in BDEBUG to contain the base sector statistics. Three locations per sector are used to store the following information:

(1) The first location containing an intersector reference (e.g., `14735)

(2) The first location available for a new reference (e.g., `14736)

(3) The last location available for a new reference (e.g., `14737)

This permits sector base tables to be initialized for all sectors contained in a 16K computer. Initialization must be in octal and be preceded by the octal sector number (e.g., `24). All sector base statistics can be specified on a single entry to BDEBUG. Failure to specify for any sector indicates that no sector base area is available in that sector. A return to the command structure is made by specifying "R" in place of a sector number.

Display Command - PDEBUG

With command routine PDEBUG the operator can selectively display the contents of the memory or the pseudoregisters. The sequencing and use of this command are given in figure 9. For illustration, the names "NL" and "PH" were defined, by using INFORM (ref. 1), to have the scale factors 2000 rpm/32 000 machine units and 10 psi/32 000 machine units, respectively.

On entry the operator specifies the initial (and final if a block) location of memory to be displayed or indicates the register to be displayed. He then enters a format number to indicate the type of display desired. Allowable format numbers are given in table XII. Format numbers 6 and 7 are available for memory display only. The interpretation of format 7 depends on the display mode selected. Two display modes are available: The print mode (selected by terminating the format number with a "SPACE") simply displays the contents of the selected cell or cells in the chosen format; the list mode (selected by terminating the format number with a "CARRIAGE RETURN") lists the selected cell or cells as illustrated in figure 4.

In the example (fig. 9), pseudoregisters A and B are first printed in octal. This is followed by scaled display of each register (initiated by "/") using direct scale-factor specification, indirect specification by referencing first "NL" and then "PH," and voltage scale-factor specification "V." Two examples of using binary scaling of the pseudoregister AREG are then given. Finally, the pseudoregister AREG is displayed in double-precision format. This causes the pseudoregisters AREG and BREG to be considered as a signed, 30-bit, double-precision integer. The attempt to display the pseudoregister BREG in double-precision format is not allowed and produces an error. Any double-precision referencing of an odd memory address also produces an error.

The remainder of the example illustrates memory display using all format numbers in the print mode and format number 7 in the list mode. Various types of address specification are used for illustration. Location M5 is displayed by first pointing to a unique name in program XXX and then the specific name M5. Once the current table

indicator has been set by this process, further specification of XXX names need not be prefixed by a unique name. Also after the first list mode example, the first address counter (pseudo program counter) is the same as the last address counter, namely X2-1. The next list mode example using first address counter +1 displays location X2.

An additional feature of PDEBUG is presented in the example. If instead of terminating a format number with a "SPACE" or "CARRIAGE RETURN" an ASCII "=" character is specified, the net result of address stringing is printed.

Modification Command - CDEBUG

With the command routine CDEBUG the operator can change the contents of memory or of any pseudoregister. The sequencing and use of this command are given in figure 10.

On entry, the first and last address counters are specified for memory modifications. The first address counter contains the first memory location to be changed. The last address counter need not be specified. If it is, it will not limit the number of memory locations that can be changed. This limit (64) is set by the length of the assembler buffer. However, if a last address is specified and the number of locations changed by the operator is less than that indicated by the difference between the last and first address counters, the remaining locations will be filled with the last assembler entry. Thus sections of memory can be filled with a constant value. The example illustrates scaled-data entry into the five locations reserved for SDAT. Since only four locations are entered, the fifth location is loaded with the fourth entry. These changes were incorporated into the program ("Y" response to "GO?").

If on entry, a pseudoregister change is specified, more than a single register can be modified (note second CDEBUG entry in example). The pseudoregisters are stored in the debug program in the following order: AREG, BREG, XREG, and KEYS (in this case `35622 to `35625, respectively). Therefore, if an AREG change is indicated, all four registers can be modified by four assembler statements. If a location beyond the pseudoregister KEYS will be modified, the operator is advised and the modification command is aborted.

The third entry into CDEBUG illustrates the assembler's handling of intersector references. The intersector reference for Y1 already existed in the sector base area of program X1. The intersector reference for Y2+1 did not exist and was therefore formed by the assembler (note listing of location `14736). These modifications were not incorporated into the program ("N" response to "GO?"). The command was aborted by specifying "R" in the location field.

Program Addition Command - ADEBUG

With the command routine ADEBUG the operator can insert additional statements into his program. The sequencing and use of this command are given in figure 11.

Specification of the first address counter indicates the location of the insertion. All additions are stored in the program addition area specified during initialization. If the original content of the insertion address is to be maintained, it must be included in the additions. (In the example listing, a jump through an intersector reference, inserted at X2-3 (`14750), is used to link the addition to the program.) The return from the addition to the program must be included in the addition by the operator.

Subroutine Specification and Execution Command - VDEBUG

The command routine VDEBUG is used for on-line structuring and execution of subroutines. These subroutines can be used for any purpose, including the on-line building of additional debug command routines. The sequencing and use of this command are given in figure 12.

On entry the first address counter specifies the location of the subroutine. The operator must then specify whether a subroutine is to be structured ("I") or executed ("X") or if a return to the command structure is desired ("R"). After subroutine structuring is completed, this specification is again required.

The usage example illustrates the structuring and execution of a routine to output the ASCII data contained in location ADAT to the teletype. The subroutine is placed starting at location Y2+1 (`15004). The statement specification illustrates the use of statement number (":") and undefined address referencing ("S") in the assembler. The first location is reserved for the return address (a "BSS" entry results in a "BSZ" response). The return from the subroutine is made in statement 13. Sector 0 is used for intersector references. Sector 15 base tables were not initialized in BDEBUG.

After the operator verifies the listing, he signals transfer of the subroutine to its permanent location ("Y" response to "GO?"). He then elects to execute the routine ("X"). The routine is executed and the characters "AB" printed. The next entry to VDEBUG illustrates execution without subroutine structuring.

Program Execution Command - XDEBUG

The command routine XDEBUG is used to execute instructions contained in the program to be debugged. The sequencing and use of this command are given in figure 13. On entry the first and last address counters are initialized. The first address counter

specifies the first location to be executed. The last address counter is an absolute execution terminator. Any instruction encountered that violates the area protected in YDEBUG will also terminate.

Other execution terminators are optionally available to the operator. He can terminate execution when an instruction is encountered that

(1) Modifies memory outside a specified override area ("M")

(2) Modifies the program counter (jump and call instructions) ("P")

(3) Causes the machine overflow latch to be set ("O")

These terminators must be set before each execution command. Any combination of the preceding can be specified. All terminators are reset whenever execution is terminated and on entry to XDEBUG.

Commands are available within XDEBUG to reset the machine overflow latch ("C") and to specify a memory override area ("A") used in conjunction with the memory-modification terminator. With this override area, a memory-modifying instruction can be executed even through the terminator is set, as long as the effective address of the instruction falls within the override area. If no override area is desired, a "-" is entered after the "A" command instead of address specification.

Two execution modes are available in XDEBUG:

(1) Single-instruction execution

(2) Execution to termination

The example illustrates the use of the terminators and execution modes. The first entry to XDEBUG demonstrates the single-instruction execution mode. Execution is specified from X1 through X2-7. The listing of the first instruction is then provided for the operator. It gives the status of the pseudoregisters (AREG = `37200). The KEYS are displayed with the four most significant bits in binary and the last six bits in octal. The remaining bits of the KEYS are meaningless. After the listing, the operator indicates single-instruction execution by the "SPACE" character. Execution terminators cannot be used in the single-instruction execution mode. The first instruction (CRA - clear AREG) is executed and the listing of the next instruction is provided. In this case it is a memory reference instruction (STA - store AREG) and therefore the effective address is also displayed (`14757). Since indexing or indirect referencing is not specified in this instruction, the effective address is the same as the direct address. The contents of the direct address before execution are also provided in the listing (fig. 4). The single-instruction execution mode is continued until a return to the command structure is specified ("R").

The second entry to XDEBUG illustrates the use of terminators. Initially, sense switch 1 is set. Execution from and to statement X1 is specified (i.e., the first and last address counters are set to X1). After the listing of the first instruction, the operator elects execution to the first program-counter modifier ("P1"). This instruction

is encountered at `14750. The operator then elects to set the memory override area to be the single location YHLD. He then specifies execution to the first memory modifier ("M↑"). This instruction is encountered at location `14747. (Incrementing the index register modifies location 0.) If he then elects to execute the program without optional termination ("↑"), he encounters a protect violation at `14753, `15001, and `15002. If he chooses to ignore these violations, he continues execution until he reaches the location specified by the contents of the last address counter (`14740). Sense switch 1 is then reset so that the instruction in location `14752 (set machine overflow latch) is encountered. Execution to overflow is specified ("O↑"). Execution is, therefore, terminated at `14752 and the next instruction is listed. The operator next resets the machine overflow latch ("C") and continues execution ("↑") until a protect violation is about to occur. He then returns to the command structure ("R").

Breakpoint Command - SDEBUG

With the command routine SDEBUG the operator can insert and delete breakpoints in the program to be debugged. Breakpoints identify the occurrence of particular program conditions. The sequencing of this command is given in figure 14. The usage example for this command is discussed in the following section (Return Command - RDEBUG).

The first address counter specifies breakpoint location. The operator then specifies either "S" to set a breakpoint or "R" to restore the original contents of the breakpoint. Five breakpoints can be inserted at any one time. Breakpoint locations and their original contents are stored in parallel address and content buffers contained in SDEBUG.

Inserting a breakpoint replaces the program instruction with a call instruction made to a reentry location in SDEBUG. The original instruction is saved. The call is made by using memory location `20 as an intersector reference. Program control is then returned to the command structure. When a breakpoint is encountered and SDEBUG is reentered, all priority interrupts are inhibited, the registers are stored in their pseudolocations, and the operator is advised of the encounter.

Return Command - RDEBUG

The command routine RDEBUG transfers control from the debug program to any other program in the machine. The sequencing and use of this command are given in figure 15.

Returns from the debug program can be made directly or through an initial condition program. On entry the operator specifies the return location by setting the first address counter and the initial-condition routine by setting the last address counter. If the last address counter is not set, no initial-condition routine is executed. The return begins when the address-counter specification is completed.

The usage example illustrates the use of both RDEBUG and SDEBUG. A breakpoint is set at location `14742. The original content of this location is displayed in octal, and control is returned to the command structure. The listing of `14742 shows the call through intersector reference location `20, which replaces the original content. The content of `20 (`25504) is the reentry location in SDEBUG. Command routine RDEBUG is then entered from the command structure, and a direct return from the debug program is specified. In program X1 the breakpoint at `14742 is encountered and indicated to the operator. Control then returns to the command structure. The operator deletes the breakpoint by using SDEBUG, and `14742 is again listed, showing its original contents to be restored. Command routine RDEBUG is again entered, and this time an initial-condition program is specified to be that subroutine programmed in the VDEBUG example. The return location is set to be X1. A call to the initial-condition routine is made ("AB" printed), and control is transferred to location X1.

Transfer Command - TDEBUG

The command routine TDEBUG transfers the contents of a block of memory to another block of memory or to an output device, fills memory from an input device, and verifies all input-output transfers. The sequencing and use of this command are given in figure 16.

On entry the address counters are initialized. These counters specify the first (e.g., XXX·SDAT) and last (XXX·SDAT+4) addresses of the memory block to be transferred or verified. In the example, the first TDEBUG entry illustrates a memory-to-memory transfer. After the memory block and transfer type ("M") are specified, the first location to receive the transfer is specified (`15040). The transfer mask, which is used to delete bits from the transferred memory word, must then be specified (`177777 for no bit modification). Any bit not set to 1 in the mask is deleted. After the transfer mask is specified, the memory transfer is made. The display command is then used to verify the transfer.

The next three entries to TDEBUG demonstrate input-output transfers and verification. In each case the input-output device number must be specified.

CONCLUSIONS

The debug program described herein offers a high level of operator-computer communication to ease the debugging process of complex assembly-language control programs. Information supplied by the computer's assembler and loader and pertinent program information supplied by the operator are used to provide communication in an easily understandable format. Communication is done through the use of identifiable names, flexible addressing, and on-line program assembly capabilities.

Eleven debug commands provide the operator with sufficient options to simplify the execution, analysis, and modification of his program and reduce program turnaround time. The program execution command contains instruction diagnostics to reduce the possibility of program destruction by program errors. Displays are available to help the operator analyze the program and to advise him of his status within the debug program. Program modifications and additions are easily incorporated on line by an extensive debug assembler. The capability to structure subroutines to aid in the debug process is also offered.

Most of the bookkeeping required in the debug process is handled by the debug program. Hard copy of the debug process is easily obtained by using the display routines. Flexibility in input-output device selection offers a versatile means of information transfer.

The command structure of the debug program allows the programmer to insert additional debug commands. The program is compatible with the data-analysis-and-display program INFORM. The ability to selectively incorporate INFORM and debug commands allows the structuring of debug and analysis programs that are tailored to meet the specific needs of the programmer.

Lewis Research Center,
National Aeronautics and Space Administration,
Cleveland, Ohio, November 20, 1978,
505-05.

APPENDIX - DESCRIPTION OF DEBUG PROGRAM

Diagrams illustrating the structure of the DEBUG program and the mechanics of its various routines are presented in this appendix. Certain parts (particularly the on-line assembler) are necessarily peculiar to the HDC-601 computer. Functional statements are used throughout so that the program can be easily adapted to other computers.

The basic input-output routines used for operator command and display are not diagrammed since they are generally straightforward and extremely machine dependent. Other routines contained in INFORM are also omitted except where necessary to the understanding of the debug program. The INFORM routines that are referenced but not diagrammed are

(1) TTYR - general-purpose number input

(2) TTYR2 - same as TTYR but with the first character prespecified by the program

(3) INPT - character input

(4) LOAD - general-purpose loader

(5) VERIFY - general-purpose verifier

(6) PUNCH - general-purpose binary dump routine

(7) CS24 - floating-point to double-precision integer conversion

These routines are described in reference 1.

The nomenclature used in the diagrams is as follows:

(1) Subroutine names are given in parentheses (e.g., (NAME))

(2) Location names are given in brackets (e.g., [CSUB])

(3) ASCII characters are given in quotation marks (e.g., "A")

(4) ASCII control characters are understored (e.g., A)

The functional statements are boxed. If more than one result can occur from execution of a statement, the results are indicated on linkage lines emanating from the statement box. All program messages are omitted from the functional diagrams for simplicity.

The command structure, given in figure 17, shows the debug command subroutines. Some of the INFORM command subroutines are also shown. See reference 1 for a complete description of INFORM's capabilities. Figures 18 to 28 are functional diagrams of the debug command subroutines. These routines are straightforward and are discussed in detail in the main body of this report. Appendix figures 29 to 39 are functional diagrams of utility routines required to support many of the debug command subroutines.

Figure 29 illustrates the structure of the on-line assembler. It consists of three parts:

(1) PTCH is used for location-field entries, to input assembler commands, and to sequence assembler events.

(2) CMDI is used for instruction- and address-field entries.

(3) ASEM is used to form the machine coding (object program) corresponding to the source-field entries.

These assembler routines require the support of additional subroutines: LIST (fig. 30), FNDI (fig. 31), FNDM (fig. 31), GSFR (fig. 32), SBSE (fig. 33), and TRAN (fig. 34).

In CMDI, two special buffers (with 64 locations each) hold the instruction- and address-field entries. If the instruction is a memory-reference instruction, the instruction op-code is stored in the instruction buffer and the positive address is stored in the address buffer. If the address is a location-field number, the second least significant bit (bit 15) of the instruction word is set. If the address is undefined, the least significant bit (bit 16) is set. Shift instructions have the op-code stored in the instruction buffer and the negative shift count stored in the address buffer. This number is always greater than `177700 (-64). All other machine instructions and pseudo-op operands are self-contained within the instruction word. Special codes are used to distinguish these instructions for the assembler and listing programs. These codes are given in table XIII.

Subroutine FNDI contains the instruction mnemonic words and their corresponding op-codes in parallel tables. This routine is used to obtain op-codes from mnemonic specification by CMDI. This routine also returns the instruction type (memory reference, shift, etc.) for address-field determinations.

Subroutine FNDM is the inverse of FNDI. A mnemonic is returned that corresponds to a specified instruction. The instruction type is indicated. This routine is used in the source listing procedure.

When the operator signals completion of source-statement entry, control is transferred to subroutine ASEM for assembly. The address buffer is interrogated to determine the instruction type. If it is a shift instruction ($-64 < \text{ADDRESS} < 0$), the shift count is combined with the op-code and stored in the corresponding assembler buffer location. If it is a memory-reference instruction ($0 \geq \text{ADDRESS}$), intersector referencing is computed by using SBSE and sector base statistics supplied by BDEBUG and modified by TRAN. If external indirect reference generation is required, the instruction counter is incremented and the external address is added to the end of the instruction buffer. The original address is appended by the indirect and index modifier bits as set in the original instruction word and stored in the corresponding assembler buffer location. Address code 8 (table XIII) is stored in the corresponding address buffer location for interrogation by the transfer routine TRAN. The original address buffer location is modified to contain the external address. The index modifier bit of the original instruction word is reset, and the indirect modifier bit is set. The instruction is then reassembled. Memory reference instructions not requiring intersector referencing are combined with the corresponding address buffer word and stored in the assembler buffer.

When assembly is completed, PTCH lists the source program by using the LIST subroutine, as well as listing all intersector reference requirements for operator verification. Verification is indicated in TRAN. If the operator is satisfied, he signals for program transfer from the assembler buffer to permanent storage. Routine TRAN does the transfer and also modifies external addresses (COD8) as required for intersector referencing. TRAN modifies the sector base statistics (see section Sector Base Statistics Command - BDEBUG) to reflect the addition of intersector references.

With the scaling routine GSFR the operator can designate scale factors to be assigned to input or output data. This routine is used for assembler decimal data entry and also in the PDEBUG command routine.

Figures 35 to 40 illustrate the debug address and name specification and handling routines. Addressing is used to specify operands for the debug commands and to satisfy the address requirements of the debug assembler.

Subroutine CMMP (fig. 35) is used for command operand specification. It initializes the first and last address counters with addresses obtained through GETN (described below). If only a single operand is specified, the address counters are set the same. Operator errors leave the address counters unaffected, and an error return from CMMP is made. Correct operand specification causes the current program indicator to be computed through CSET (fig. 36).

In subroutine CSET, the contents of the first address counter are compared with the relocation base of the various address tables. The maximum base value that is less than the contents of the first address counter is determined, and the address of the corresponding primary-name table is stored in a location reserved for the current program indicator.

Subroutines GETN and ADRS (fig. 37) initialize the address specification subroutine GETA. Subroutine GETN specifies the debug name table; subroutine ADRS specifies the INFORM name table and prohibits address indexing and the special assembler address formats. Subroutine CMCK (fig. 37) flags special assembler address formats as errors when warranted.

Subroutine GETA (fig. 38) is a general-purpose operator address specification routine. It is used for all addressing in both INFORM and the debug program. Initialization and entry must be made through GETN or ADRS. Certain implicit initializations are also required. The operator is allowed to reference a special external-bias address set through an INFORM command. Referencing of this address is allowed only after the address stringing delimiters (see below). The first address counter is used by address displacement delimiters (see below) as the reference address. It is generally set when specifying command operands in CMMP but may also be set in the ADEBUG, PDEBUG, and XDEBUG command routines (see the section DEBUG COMMANDS).

On entry to GETA, a versatile address input format is available to the operator through the use of address delimiters and postaddress delimiters. The NAME subroutine (fig. 39) specifies either a name or an address delimiter. If a name is entered, the name table specified by GETN or ADRS is searched. If the name is found, the corresponding address is determined. The termination character of the name entry specifies the postaddress delimiter.

Subroutine NAME is used for operator name specification. The primary and secondary name words are formed and made common to other routines. Any character other than an alphanumeric, a delimiter, will terminate the name entry. The terminating character is tested to insure proper name entry in the calling routines. If no name is entered, the name words are filled with "SPACE" characters, and the delimiter entered is treated as a command by the calling routines. This subroutine is used to input most INFORM and debug names.

Subroutine NFIN (fig. 40) locates a name assigned to a specified address. All name tables are searched. If the address is located, the corresponding name words are returned. If the address cannot be located, NFIN returns name words filled with "SPACE" characters.

Subroutine FIND (fig. 40) obtains the displacement of a specified name from the start of a name table. On entry the table to be searched and the displacement between the primary and secondary name tables must be specified (since FIND can interrogate INFORM tables also). An error exit is used if the name cannot be found. The displacement is used by the calling routine to determine the address assigned to the name.

REFERENCES

1. Cwynar, David S.: INFORM - An Interactive Data Collection and Display Program with Debugging Capability. NASA TP-1424, 1979.

2. HDC-601 Digital Computer Programmers Reference Manual. Honeywell Aerospace Division, St. Petersburg, Fla, Apr. 1971.

3. HDC-601 Digital Computer Software Systems Description. Honeywell Aerospace Division, St. Petersburg, Fla., Aug. 1972.

TABLE I. - ALLOWABLE ADDRESS DELIMITERS

Address delimiter	Description
APOSTROPHE	Indicates that absolute octal entry follows
*	Specifies first address counter as entry
/	Indicates that relative octal entry follows
SPACE	Specifies first address counter plus 1 as entry
$	Indicates that an undefined assembler address follows
:	Indicates that a defined assembler address follows

TABLE II. - ALLOWABLE POSTADDRESS DELIMITERS

Postaddress delimiter	Description
+	Indicates address addition
-	Indicates address subtraction
,	Assembler address indexing (must be followed by SPACE)
SPACE	Signals completion of address entry
.	Specifies a unique name

TABLE III. - ALLOWABLE ASSEMBLER LOCATION-FIELD ENTRIES

Location-field entry	Description
n	Indicates location of following instruction in operators program (n = decimal integer, $0 \leq n \leq 63$)
$n	Indicates that following location defines undefined address, $n (n = decimal integer, $0 \leq n \leq 63$)
\	Indicates completion of operator's instruction entries
M	Allows operator to view and modify the location counter
R	Returns to INFORM/debug command structure

TABLE VI. - ALLOWABLE SCALE-FACTOR ENTRIES

Scale-factor entry	Description
E.U./M.U.	Directly specifies ratio of engineering unit (E.U.) to machine unit (M.U.)
NAME	Specifies scale factor assigned to a name in INFORM table
V	Specifies voltage scaling (10 volts/32 000 M.U.)
*	Defaults to last entered scale factor
B(nn)	Specifies binary scaling, where (nn) represents any + or - integer. (2^{nn} E.U./32 768 M.U.)

TABLE V. - ALLOWABLE PSEUDO-OPERATION MNEMONICS

Mnemonic	Address-field entry
OCT	Octal integer data
DEC	Decimal integer data
FPC	Decimal floating-point data
BCI	ASCII data
DPC	Decimal double-precision integer data
DAC	Direct-address constant
BSS	Block-storage reservation
BSZ	Block-zero storage reservation

TABLE IV. - ALLOWABLE ASSEMBLER INSTRUCTION-FIELD ENTRIES

Instruction-field entry	Description
APOSTROPHE	Indicates entry of an instruction or datum coded in octal (The OCT pseudo-operation may also be used for octal data entry.)
XXX	Instruction mnemonic (where XXX is any machine instruction mnemonic)
YYY	Pseudo-operation mnemonic (where YYY is any pseudo-operation defined in table VIII)
*	Specifies indirect addressing (also signals completion of instruction-field entry)
SPACE	Signals completion of instruction-field entry

TABLE VII. - DEBUG COMMAND ROUTINES

Routine	Entry command[a]	Figure	Description
YDEBUG	Y	18	Initializes debug and INFORM programs
UDEBUG	U	19	Assigns debug source names and addresses
BDEBUG	B	20	Initializes sector base statistics
PDEBUG	P	21	Displays memory or register
CDEBUG	C	22	Modifies program or pseudoregister
ADEBUG	A	23	Inserts additional statements into program
VDEBUG	V	24	Specifies and executes subroutine
XDEBUG	X	25	Executes program
SDEBUG	S	26	Inserts or deletes breakpoint
RDEBUG	R	27	Returns to program
TDEBUG	T	28	Transfers memory from block to block

[a]Underscoring denotes ASCII control character.

TABLE VIII. - DEBUG ERROR MESSAGES

Error message	Meaning	Program control transfer
?	Nonrecoverable error	Command structure
X?	Execution command error	XDEBUG command input and decoder
NOT EX	Nonexecutable instruction encountered in XDEBUG	Command structure
CSET	Last instruction executed in XDEBUG caused overflow latch to be set	None
PV	Execution of following instruction will cause protect violation	None
OVFLON	Overflow of debug name tables using UDEBUG	Command structure
OVFLOP	Overflow of program addition area using ADEBUG	Debug assembler's reentry location
FORMAT	Assembler format error	
NO $ REF	Operator tried to define a nonexistent undefined address reference	
$n	Operator failed to define undefined address reference $n	
OVFLOM	Overflow of assembler's buffer or instrument buffer (64 locations)	
OVFLOB	No intersector reference locations are available	None
SPOF	Single-precision overflow	Command structure
$N	Name specification error	Start of NAME
OVFLOA	Specified address is negative	Command structure
IDR>32	Indefinite indirect chain encountered in XDEBUG	None
S?	SDEBUG error	Command structure

TABLE IX. - SEQUENCING FOR ADDRESS-FIELD SPECIFICATION

Step	Enter	Terminate	Response	Go to step	Description
1a	(Name)		None	1	Temporarily specifies name table that contains entry
1b	(Name)	None		2	Specifies name in specified name table
1c	(Octal)				Specifies absolute octal address
1d	/(Octal)				Specifies relative octal address
1e	*				Specifies first address counter as address
1f	SPACE				Specifies first address counter plus 1 as address
2a	SPACE			Exit	Terminates address entry; returns to calling program
2b	+			1	Indicates address addition
2c	-			1	Indicates address subtraction
2d	,		1	3	Specifies address indexing
3	SPACE		None	Exit	Terminates address entry; returns to calling program

TABLE X. - SEQUENCING FOR SCALE-FACTOR SPECIFICATION

Step	Enter	Terminate	Response	Go to step	Description
--	---------	------	SF=	1	Advises operator of scale-factor entry requirement
1a	(Decimal)	SPACE	/	2	Specifies engineering units (numerator)
1b	+	None	None	Exit	Specifies last scale-factor entry as scale factor
1c	V	None		Exit	Specifies 10 volts/32 000 machine units as scale factor
1d	-	None		3	Indicates use of INFORM name for specification
1e	Bnn	SPACE		Exit	Specifies $2^{nn}/32\,768$ as scale factor
2	(Decimal)	SPACE		Exit	Specifies machine units (denominator)
3	(Name)	SPACE		Exit	Uses scale factor specified for INFORM name

TABLE XI. - SEQUENCING FOR ON-LINE ASSEMBLER

Step	Enter	Terminate	Response	Go to step	Description
1a	R	None	None	Return	Terminates command and return to command structure
1b	\	None	Listing	Exit	Indicates end of entries. Returns to calling program
1c	M	None	Output	2	Advises operator of location count and permit change
1d	(Decimal)	SPACE	Spaces	3	Specifies location-field entry number
1e	$(Decimal)	SPACE	CRLF	1	Specifies undefined address definition
2a	,	None	↓	↓	Indicates no change in location count
2b	(Decimal)	SPACE	↓	↓	Specifies location count
3a	(Octal)	SPACE	↓	↓	Octal data and instruction entry
3b	(Mnemonic)	SPACE	Spaces	4	Mnemonic entry requiring address
3c	↓	*	Spaces	4	Mnemonic entry requiring indirect address
3d	↓	SPACE	Spaces	5	Mnemonic entry requiring operand
3e	↓	SPACE	CRLF	1	Mnemonic entry not requiring address or operand
4a	Address	SPACE	CRLF	↓	Specifies address
4b	Address	,	1 CRLF	↓	Specifies address with indexing
4c	(Decimal)	SPACE	CRLF	↓	Specifies location number as address
4d	(Decimal)	,	1 CRLF	↓	Specifies location number as address with indexing
4e	$(Decimal)	SPACE	CRLF	↓	Specifies undefined address
4f	$(Decimal)	,	1 CRLF	↓	Specifies undefined address with indexing
5a	(Number)	SPACE	CRLF	↓	Specifies operand
5b	(Number)	/	SF=	↓	Specifies sealed operand (used with decimal-entry pseudo-operations only)

TABLE XII. - FORMAT NUMBERS FOR DISPLAY COMMAND

Format number	Display type
0	Six-digit octal
1	Decimal integer
2	Decimal floating point
3	ASCII
4	Double-precision decimal integer
5	Direct-address constant
[a]6	Block storage (displays number of successive zero locations)
[a]7	Effective-address contents (print mode)
[a]7	Machine instruction (list mode)
/	Scaled floating-point decimal (see section Scaling and table X)

[a]Available for memory display only.

TABLE XIII. - ASSEMBLER ADDRESS-FIELD CODING

Code	Octal value	Description
0	177000	Specifies octal integer data
1	177001	Specifies decimal integer data
2	177002	Specified decimal floating-point data
3	177003	Specified ASCII data
4	177004	Specifies double-precision integer data
5	177005	Specifies direct-address constant
6	177006	Specifies block-storage reservation
7	177007	Specifies nonmemory reference instructions
8	177010	Specifies external address constants required by assembler for intersector referencing

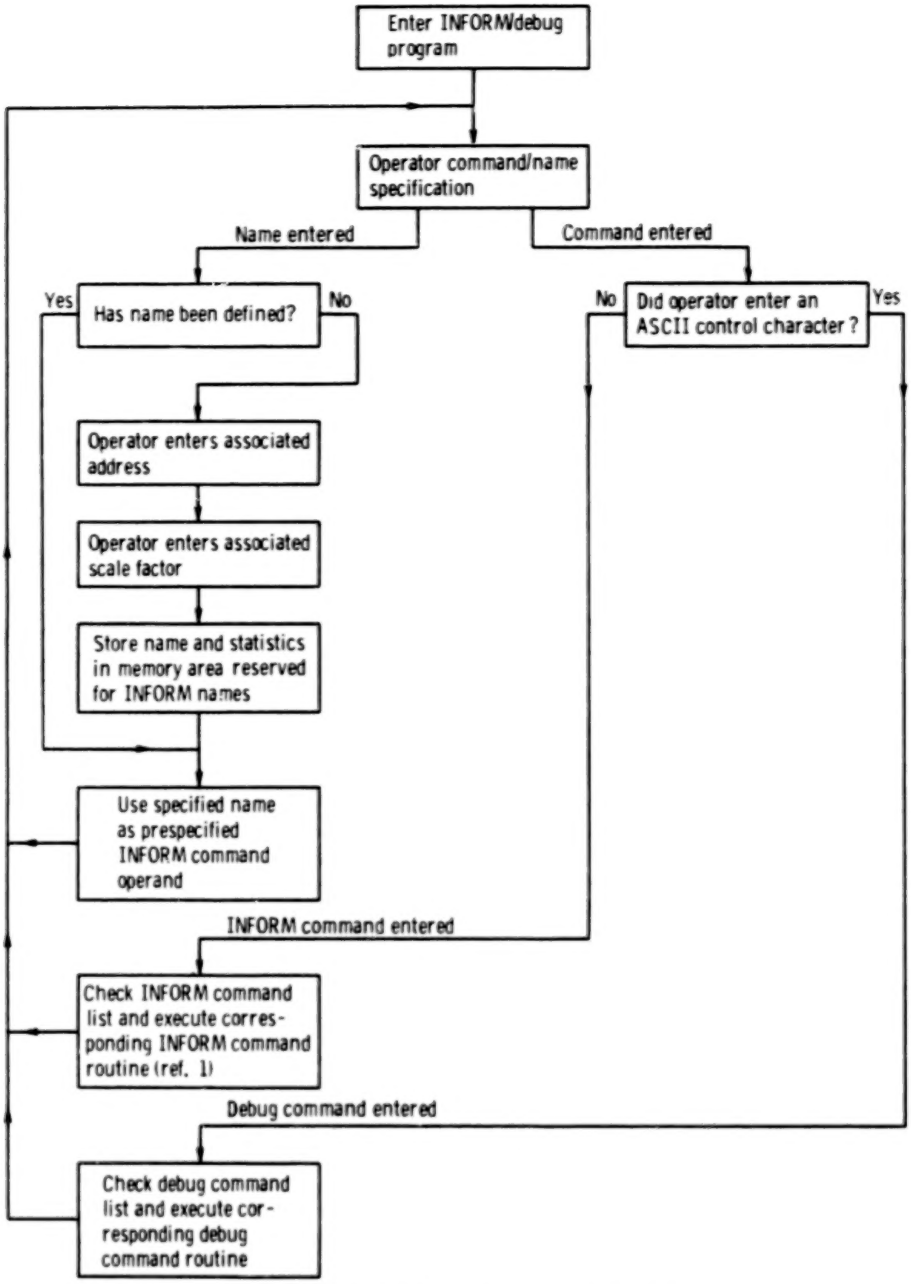

Enter INFORM/debug program
Operator command/name specification
Name entered
Command entered
Yes
Has name been defined?
No
No
Did operator enter an ASCII control character ?
Yes
Operator enters associated address
Operator enters associated scale factor
Store name and statistics in memory area reserved for INFORM names
Use specified name as prespecified INFORM command operand
INFORM command entered
Check INFORM command list and execute corresponding INFORM command routine (ref. 1)
Debug command entered
Check debug command list and execute corresponding debug command routine

Figure 1. - Functional diagram of command structure.

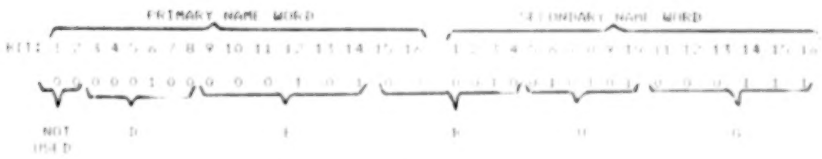


Figure 2. - Character packing for name words.

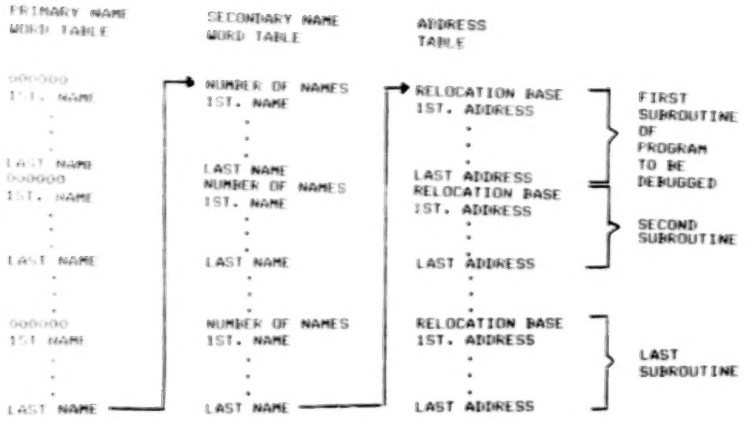


Figure 3. - Format of debug name and address tables.

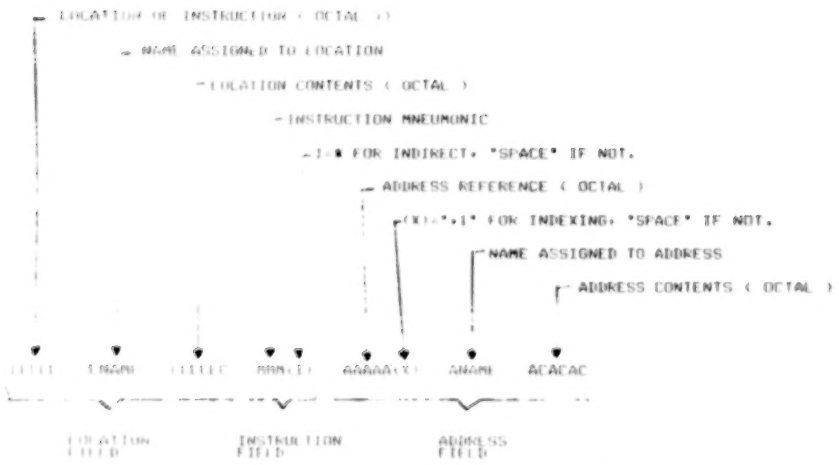


Figure 4. - Source listing format for memory reference instructions.

Octal location	Location contents	Location name	Instruction field	Address field	Comments
14735	025000				1st location used for intersector reference
14736	000000				1st location available for intersector reference
14737	000000				Last location available for intersector reference
14740	140040	X1	CRA		Program start. Clear A register.
14741	011757		STA	YHLD	Store A register in 14757
14742	073760		LDX	M5	Load X register with -5
14743	105761		LDA*	YDAC	Load A register indirect from 14761
14744	023757		CAS	YHLD	Compare A register to YHLD
14745	011757		STA	YHLD	Here if > YHLD, store A register in YHLD
14746	101000		NOP		Here if = YHLD, no operation
14747	024000		IRS	0	Here if < YHLD, increment X register. Skip if zero
14750	003743		JMP	*-5	Jump to 14743
14751	101020		SS1		Skip if sense switch 1 set
14752	140600		SCB		Set overflow latch
14753	121735	X2	CALL	YYY	Call program YYY through indirect reference
14754	014757		DAC	YHLD	Direct address constant
14755	154776		DAC*	YHLD, 1	Direct address constant, indexed, indirect
14756	003740		JMP	X1	Jump to 14740
14757	000000	YHLD	BSZ	1	One location reserved for YHLD
14760	177773	M5	DEC	-5	Decimal integer
14761	054775	YDAC	DAC	EDAT, 1	Direct address constant, indexed
14762	140702	ADAT	BCI	2, ABC	ASCII data
14763	141640				
14764	037346	FDAT	DEC	0.1	Floating point
14765	063146				
14766	000000	DDAT	DPC	4	Double precision
14767	000004				
14770	000000	SDAT	BSZ	5	Five locations reserved for SDAT
14771	000000				
14772	000000				
14773	000000				
14774	000000				
14775	177777	EDAT	OCT	177777	Octal integer
14776	000000	XHLD	BSZ	1	One location reserved for XHLD
15000	000000	Y1	BSZ	1	Start of program YYY
15001	025000		IRS	Y1	Increment Y1 (return address)
15002	025000		IRS	Y1	Again
15003	103000	Y2	JMP*	Y1	Return to calling program

Figure 5. - Example program for debug illustrations.

Step	Enter	Terminate	Response	Go to step-	Description
1	Y	-----	Y: IDP?	2	Enters YDEBUG from command structure
2a	I	-----	START=	3	Indicates INFORM initialization
2b	D	-----	PATCH=	5	Indicates debug initialization
2c	P	-----	=	9	Indicates protect-area initialization
3	Octal	SPACE	NUMBER=	4	Specifies start address of INFORM name tables
4	|	|	Display	Return	Specifies maximum number of INFORM names
5	|	|	ASMBL=	6	Specifies start address of program addition area
6	|	|	NTABL=	7	Specifies start address of assembler buffer
7a	↓	↓	NUMBER=	8	Specifies start address of debug name tables
7b	#	-----	----------	Return	Defaults debug name table statistics
8	Octal	SPACE	Display	Return	Specifies maximum number of debug names
9a	Octal	SPACE	←	10	Specifies first location of protect area
9b	#	-----	----------	Return	Specifies no area to be protected
10	Octal	SPACE	----------	Return	Specifies last location of protect area

(a) Sequencing.

```
←Y: IDP? I
START='16000
NUMBR='24
 '016426
←Y: IDP? D
PATCH='16426
ASMBL='16700
NTABL='17000
NUMBR='24
 '017074
←Y: IDP? PP= '15000 ←'15003
←
```

(b) Example.

Figure 6. - Sequencing and use of YDEBUG command routine.

Step	Enter	Terminate	Response	Go to step-	Description
1	U	----------	U: BASE=	2	Enters UDEBUG from command structure.
2	Octal	CARRIAGE RETURN	----------	3	Specifies relocation base for source-name entries
3a	Name	SPACE	----------	4	Specifies a source name
3b	:	----------	----------	6	Indicates change of input device
3c	*Comment	CARRIAGE RETURN	----------	3	Allows program to ignore comment lines
4a	W	----------	Display	Return	Terminates current program source-name entry
4b	Zero	----------	----------	5	Indicates address entry
5a	Octal	CARRIAGE RETURN	----------	3	Specifies relative address of source name
5b	|	SPACE	----------	|	Specifies relative address of source name
5c	|	A	----------	|	Specifies absolute address of source name
6	↓	CARRIAGE RETURN	----------	↓	Specifies input device number

(a) Sequencing.

```
←U:  BASE='14740

XXX   00
X1    00
X2    013
YHLD  017
M5    020
YDAC  021
ADAT  022
FDAT  024
DDAT  026
SDAT  030
EDAT  014775A
XHLD  036
0000  #
 '000015
←U:  BASE='15000

YYY   00
Y1    00
Y2    03
0000  #
 '000021
←
```

(b) Example.

Figure 7. - Sequencing and use of UDEBUG command routine.

Step	Enter	Terminate	Response	Go to step-	Description
1	B	-----	B:	2	Enters BDEBUG from command structure
2a	R	-----	----	Return	Terminates entries
2b	Octal	SPACE	----	3	Specifies sector number
3a	R	-----	----	Return	Terminates entries
3b	Octal	SPACE	----	4	Specifies first location used for intersector reference
4a	R	-----	----	Return	Terminates entries
4b	Octal	SPACE	----	5	Specifies first location available for intersector reference
5a	R	-----	----	Return	Terminates entries
5b	Octal	SPACE	----	2	Specifies last location available for intersector reference

(a) Sequencing.

```
←B:
'0 '600 '600 '700
'14 '14735 '14736 '14737
'4
←
```

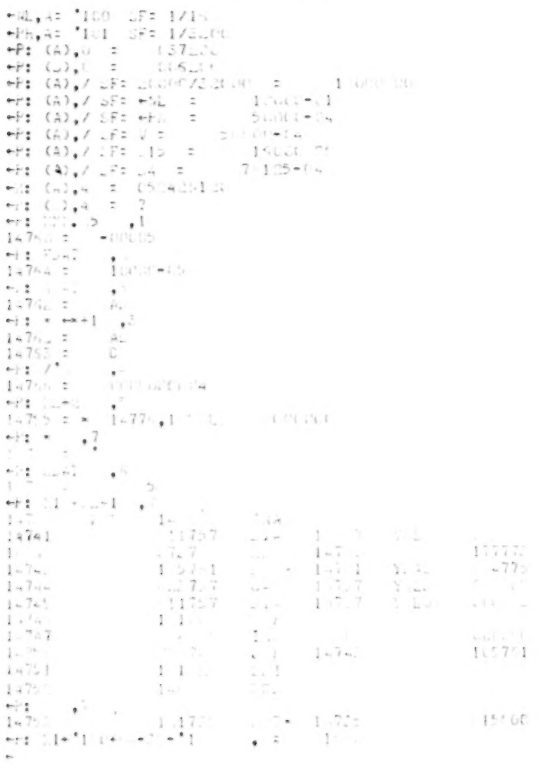
(b) Example.

Figure 8. - Sequencing and use of BDEBUG command routine.

Step	Enter	Terminate	Response	Go to step-	Description
1	P	-----	P:	2	Enters PDEBUG from command structure
2a	(A	-----	),	5	Specifies display of pseudo A register
2b	(B	-----	),	↓	Specifies display of pseudo B register
2c	(X	-----	),	↓	Specifies display of pseudo X register
2d	(K	-----	),	↓	Specifies display of pseudo K register
2e	Address[a]	SPACE	-----	3	Specifies first address of memory to be displayed
3a	SPACE	-----	,	5	Indicates single display
3b	←	-----	-----	4	Indicates multiple display
4	Address[a]	SPACE	,	5	Specifies last address of memory to be displayed
5	Decimal[b]	SPACE	-----	6	Specifies display format
6a	SPACE	-----	Display	Return	Displays specified entry in print mode
6b	CARRIAGE RETURN[c]	-----	Display	Return	Displays specified entry in list mode
6c	=[c]	-----	Display	Return	Displays memory address

[a]See table IX.
[b]See tables X and XII.
[c]Not allowed for register display.

(a) Sequencing.



(b) Example.

Figure 9. - Sequencing and use of PDEBUG command routine.

Microfilmed From
Best Available Copy

Step	Enter	Terminate	Response	Go to step-	Description
1	C	-----	C:	2	Enters CDEBUG from command structure
2a	(A	-----	------	5	Specifies modification of pseudo A register
2b	(B	-----	------	↓	Specifies modification of pseudo B register
2c	(X	-----	------	↓	Specifies modification of pseudo X register
2d	(K	-----	------	↓	Specifies modification of pseudo K register
2e	Address[a]	SPACE	------	3	Specifies first address of memory to be modified
3a	SPACE	-----	------	5	Sets last address counter to first address
3b	←	-----	------	4	Indicates memory area to be modified
4	Address[a]	SPACE	------	5	Specifies last address of memory to be modified
5	Source[b]	\	Listing	6	Enters assembler for modification specification
6a	Y	-----	------	Return	Incorporates modifications
6b	N	-----	------	5	Reenters assembler for changes

[a]See table IX.
[b]See table XI.

(a) Sequencing.

(b) Example.

Figure 10. - Sequencing and use of CDEBUG command routine.

Step	Enter	Terminate	Response	Go to step-	Description
1	A	-----	A:	2	Enters ADEBUG from command structure
2	Address[a]	SPACE	------	3	Specifies address of program addition
3	Source[b]		Listing	4	Enters assembler for addition specification
4a	Y	-----	------	Return	Inserts addition into program
4b	N	-----	------	3	Reenters assembler for changes

[a]See table IX.
[b]See table XI.

(a) Sequencing.

(b) Example.

Figure 11. - Sequencing and use of ADEBUG command routine.

Step	Enter	Terminate	Response	Go to step-	Description
1	V	-----	V:IXR?	2	Enters VDEBUG from command structure
2	Address[a]	SPACE	-------	3	Specifies subroutine address
3a	I	-----	-------	4	Specifies that subroutine is to be programmed
3b	X	-----	-------	Return	Specifies that subroutine is to be executed
3c	R	-----	-------	Return	Specifies return to command structure
4	Source[b]	\	Listing	5	Enters assembler for subroutine specification
5a	Y	-----	-------	3	Transfers subroutine to specified location
5b	N	-----	-------	4	Reenters assembler for changes

[a]See table IX.
[b]See table XI.

(a) Sequencing.

```
←V: YYY.Y2+1
IXR? I

  0    BSS   1     00000
  1    LDX   $1    00001
  2    LDA   XXX.ADAT    00002
  3    IAB         00003
  4    CRA         00004
  5    LLL   8     00005
  6    SKS   'I04        00006
  7    JMP   :6    00007
  8    OCP   'I04        00008
  9    OIA   '4    00009
  10   JMP   :8    0001C
  11   IAS   'O    00011
  12   JMP   :4    00012
  13   JMP*  :C    00013
  $1
  14   DEC   -2    00014
  \

15004          000000   BSZ   00001
15005          073022   LDX   15022          121726
15006          104600   LDA*  00600          000000
15007          000201   IAB
15010          140140   CRA
15011          041070   LLL   08
15012          070104   SKS   0104
15013          013012   JMP   15012          015013
15014          030104   OCP   0104
15015          170004   OIA   0004
15016          003014   JMP   15014          103725
15017          024000   IAS   00000          000000
15020          003010   JMP   15010          121725
15021          103004   JMP*  15004          111716
15022          177776   DEC   -00002
00600          014762   DAC   14762   ADAT   140702
?0? Y
IXR? XA
←V: Y2+1
IXR? XA
←
```

(b) Example.

Figure 12. - Sequencing and use of VDEBUG command routine.

Step	Enter	Terminate	Response	Go to step-	Description
1	X	-----	X:	2	Enters X DEBUG from command structure
2	Address[a]	SPACE	------	3	Specifies first address to be executed
3a	SPACE	-----	Listing	5	Sets execution stop address to first address
3b	←	-----	------	4	Indicates stop address to be specified
4	Address[a]	SPACE	Listing	5	Specifies execution stop address
5a	C	-----	------	↓	Clears overflow latch (bit 1 of pseudo K register)
5b	M	-----	------	↓	Sets memory modifier terminator
5c	P	-----	------	↓	Sets program counter modifier terminator
5d	O	-----	------	↓	Sets overflow terminator
5e	A	-----	:	6	Indicates memory-modifier termination to be qualified
5f	.	-----	Listing	5	Executes to stop address or as indicated by terminators
5g	SPACE	-----	Listing	5	Executes single instruction (terminators not applicable)
5h	R	-----	------	Return	Returns to command structure
6a	Address[a]	SPACE	------	7	Specifies first address that can be modified
6b	—	-----	------	5	Specifies memory-modifier termination not qualified
7a	SPACE	-----	------	5	Indicates that only single location can be modified
7b	←	-----	------	8	Indicates that memory area can be modified
8	Address[a]	SPACE	------	5	Specifies last address in area that can be modified

[a]See table IX.

(a) Sequencing.

```
←/: XXX.X1 ←XXX.X2+7

   037200A   006200B   177773X   0001K00
14740   XXX     140040   CBA

   000000A   006200B   177773X   0001K00       014757EFA
14741           011757   STA   14757   YHLD     000000

   000000A   006200B   177773X   0001K00       014760EFA
14742           073760   LDX   14760   M5       177773

   000000A   006200B   177773X   0001K00       014770EFA
14743           105761   LDA*  14761   YDAC     054775

   174700A   006200B   177773X   0001K00       014757EFA
14744 *         023757   CAS   14757   YHLD     000000

   174700A   006200B   177773X   0001K00       000000EFA
14747           024000   IXS   00000           000000

/
←
```

(b) Example.

Figure 13. - Sequencing and use of XDEBUG command routine.

MICROFILMED FROM BEST AVAILABLE COPY

```
←X: X1

   174700A   006200B   177773X   0001K00
14740    XXX     140040   CmA

P↑
   174700A   006200B   177774C   0001K00       014743EFA
14750            003743   JMP   14743           105761

AP: YNLD
M↑
   006200A   006200B   177774D   0001K00       015000EFA
14747            024000   IAS   00000           000000

↑
PV
   014400A   006200B   000000X   0001K00       015000EFA
14753    X2      121735   JST*  14735           015000

↑
PV
   014400A   006200B   000000X   0001K00       015000EFA
15001            025000   IAS   15000    YYY    014754

↑
PV
   014400A   006200B   000000X   0001K00       015000EFA
15002            025000   IAS   15000    YYY    014755

↑
   014400A   006200B   000000X   0001K00
14740    XXX     140040   CmA

(NOTE: AT THIS POINT SENSE SWITCH 1 IS RESET)

O↑
C SET   014752
PV
   014400A   006200B   000000X   1001K00       015000EFA
14753    X2      121735   JST*  14735           015000

C↑
PV
   014400A   006200B   000000X   0001K00       015000EFA
15001            025000   IAS   15000    YYY    014754

R
←
```

(b) Concluded.

Figure 13. - Concluded.

Step	Enter	Terminate	Response	Go to step-	Description
1	S	-----	S:	2	Enters SDEBUG from command structure
2	Address[a]	SPACE	-------	3	Specifies breakpoint location
3	SPACE	-----	SR?	4	Options indicated
4a	S	-----	Display	Return	Specifies set-breakpoint option
4b	R	-----	Display	Return	Specifies delete-breakpoint option

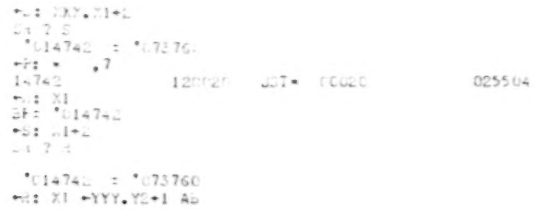
[a]See table IX.

Figure 14. - Sequencing of SDEBUG command routine.

Step	Enter	Terminate	Response	Go to step-	Description
1	R	------	R:	2	Enters RDEBUG from command structure
2	Address[a]	SPACE	------	3	Specifies return address
3a	←	------	------	4	Indicates initial-condition routine to be executed
3b	SPACE	------	------	5	Indicates a direct return
4	Address[a]	SPACE	------	5	Calls routine specified by entry
5	--------	------	Return	---	Transfers program control to return address

[a]See table IX.

(a) Sequencing.



(b) Example.

Figure 15. - Sequencing and use of RDEBUG command routine.

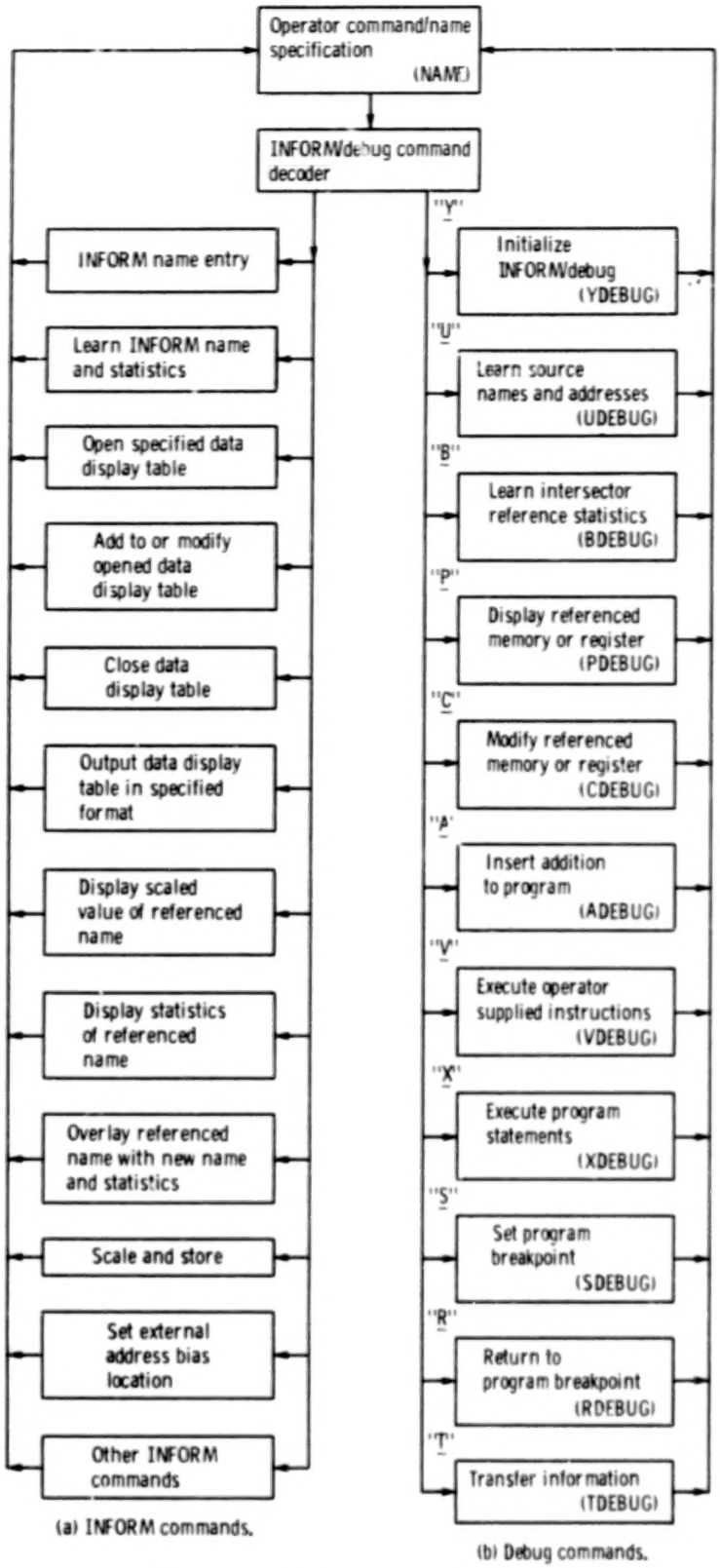
Step	Enter	Terminate	Response	Go to step-	Description
1	I	-----	T:	2	Enters TDEBUG from command structure
2	Address[a]	SPACE	------	3	Specifies first address of memory to be transferred
3a	SPACE	-----	------	5	Transfers only single location
3b	←	-----	------	4	Indicates transfer of a memory area
4	Address[a]	SPACE	------	5	Specifies last address of memory to be transferred
5a	P	-----	UNIT=	9	Indicates transfer to output device
5b	V	-----	UNIT=	9	Indicates verification against input device
5c	L	-----	UNIT=	9	Indicates transfer from input device
5d	M	-----	T:	6	Indicates transfer to memory
6	Address[a]	SPACE	------	7	Specifies start of location to receive transfer
7	SPACE	-----	MASK=	8	Indicates no specification of last-address counter
8	Octal	SPACE	------	Return	Specifies six-digit octal mask and transfer
9	Octal	SPACE	------	Return	Specifies octal device number and transfer or verify

[a]See table IX.

(a) Sequencing.

(b) Example.

Figure 16. - Sequencing and use of TDEBUG command routine.



(a) INFORM commands.

(b) Debug commands.

Figure 17. - INFORM/debug command structure.

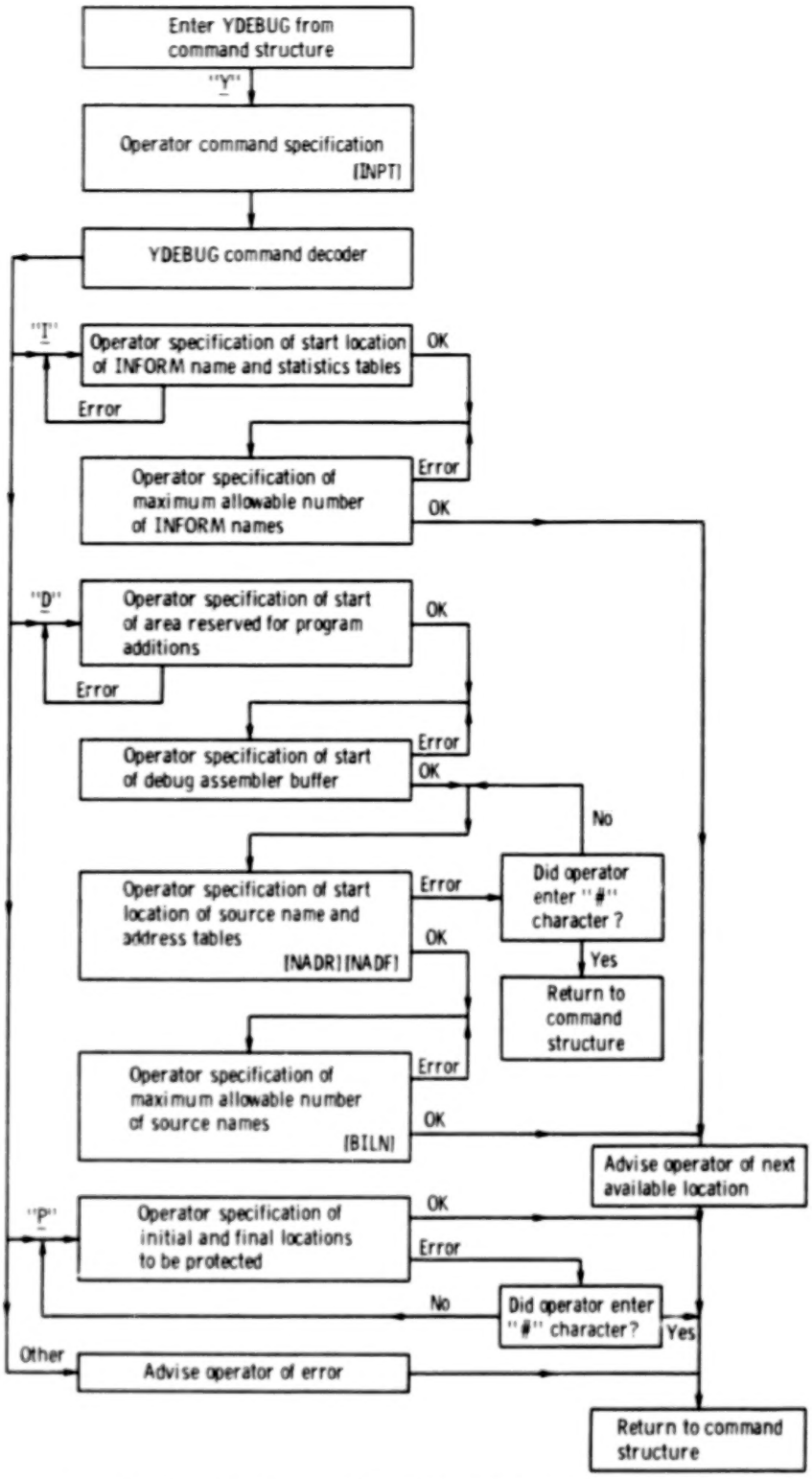


Figure 18. - Functional diagram of INFORM/debug initialization command (YDEBUG).

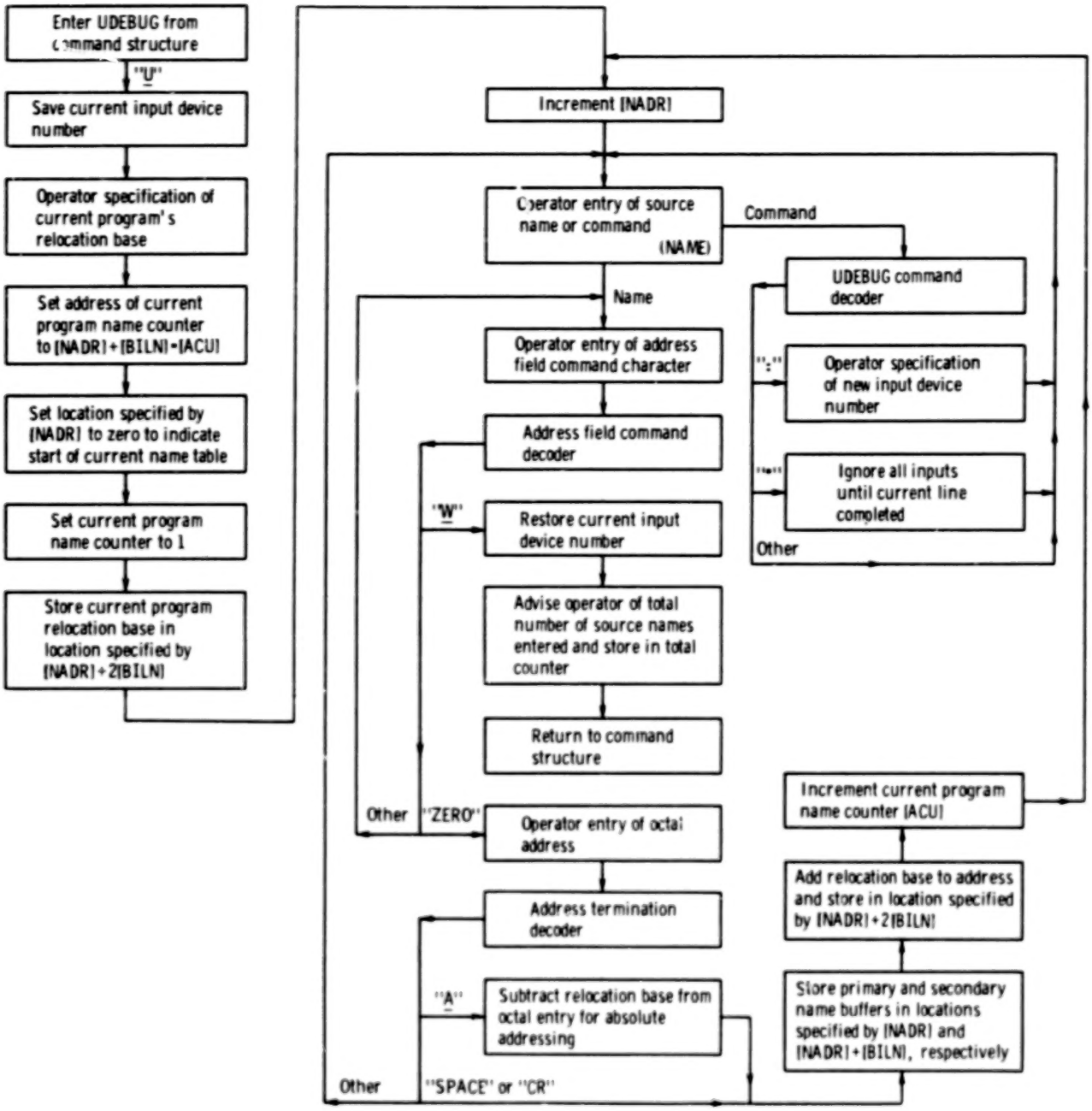

Enter UDEBUG from command structure
"U"
Save current input device number
Operator specification of current program's relocation base
Set address of current program name counter to [NADR]+[BILN]+[ACU]
Set location specified by [NADR] to zero to indicate start of current name table
Set current program name counter to 1
Store current program relocation base in location specified by [NADR]+2[BILN]
Increment [NADR]
Operator entry of source name or command (NAME)
Command
Name
UDEBUG command decoder
Operator entry of address field command character
":"
Operator specification of new input device number
Address field command decoder
"*"
Ignore all inputs until current line completed
Other
"W"
Restore current input device number
Advise operator of total number of source names entered and store in total counter
Return to command structure
Other
"ZERO"
Operator entry of octal address
Address termination decoder
"A"
Subtract relocation base from octal entry for absolute addressing
Other
"SPACE" or "CR"
Increment current program name counter [ACU]
Add relocation base to address and store in location specified by [NADR]+2[BILN]
Store primary and secondary name buffers in locations specified by [NADR] and [NADR]+[BILN], respectively

Figure 19. - Functional diagram of source-name table input command (UDEBUG).

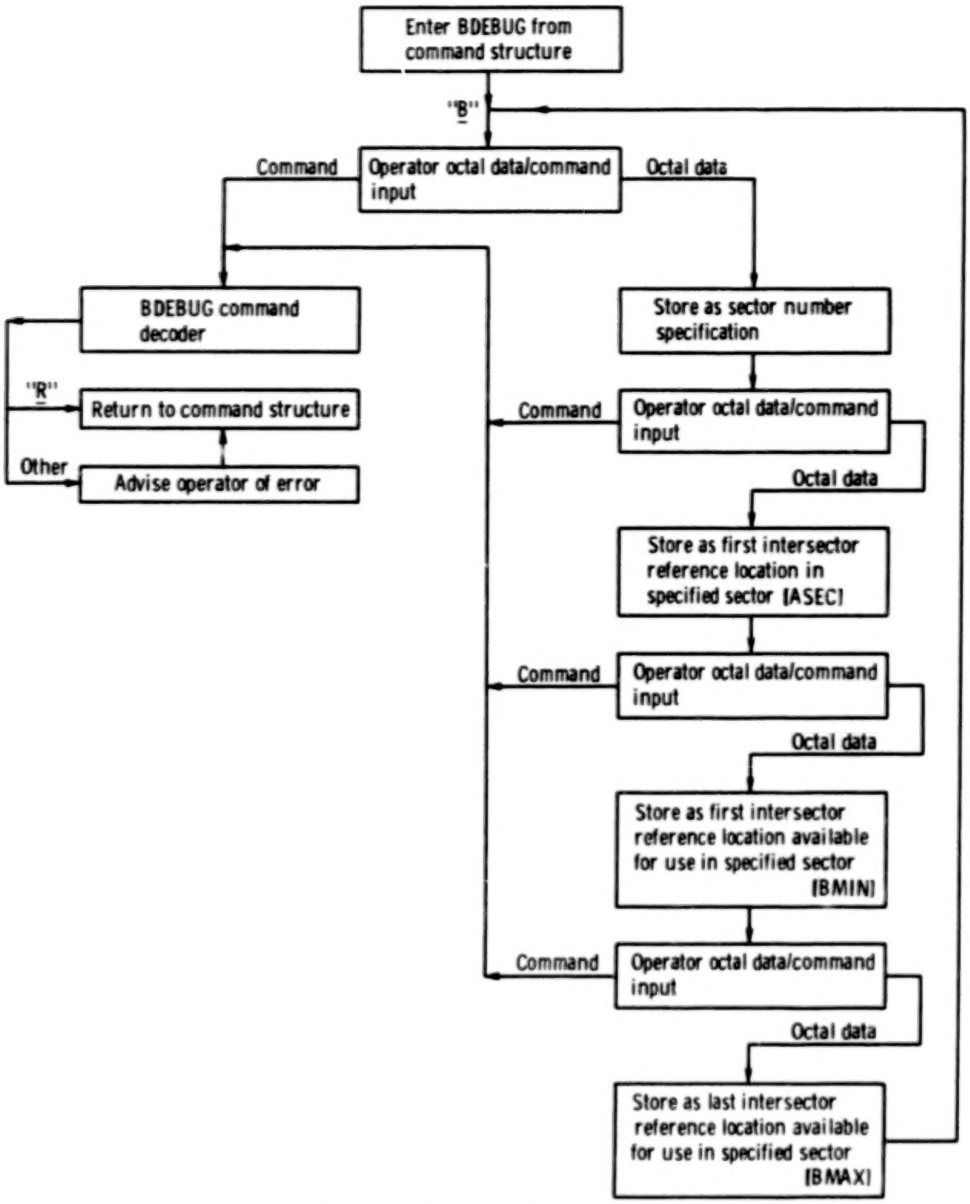


Figure 20. - Functional diagram of intersector reference statistics input command (BDEBUG).

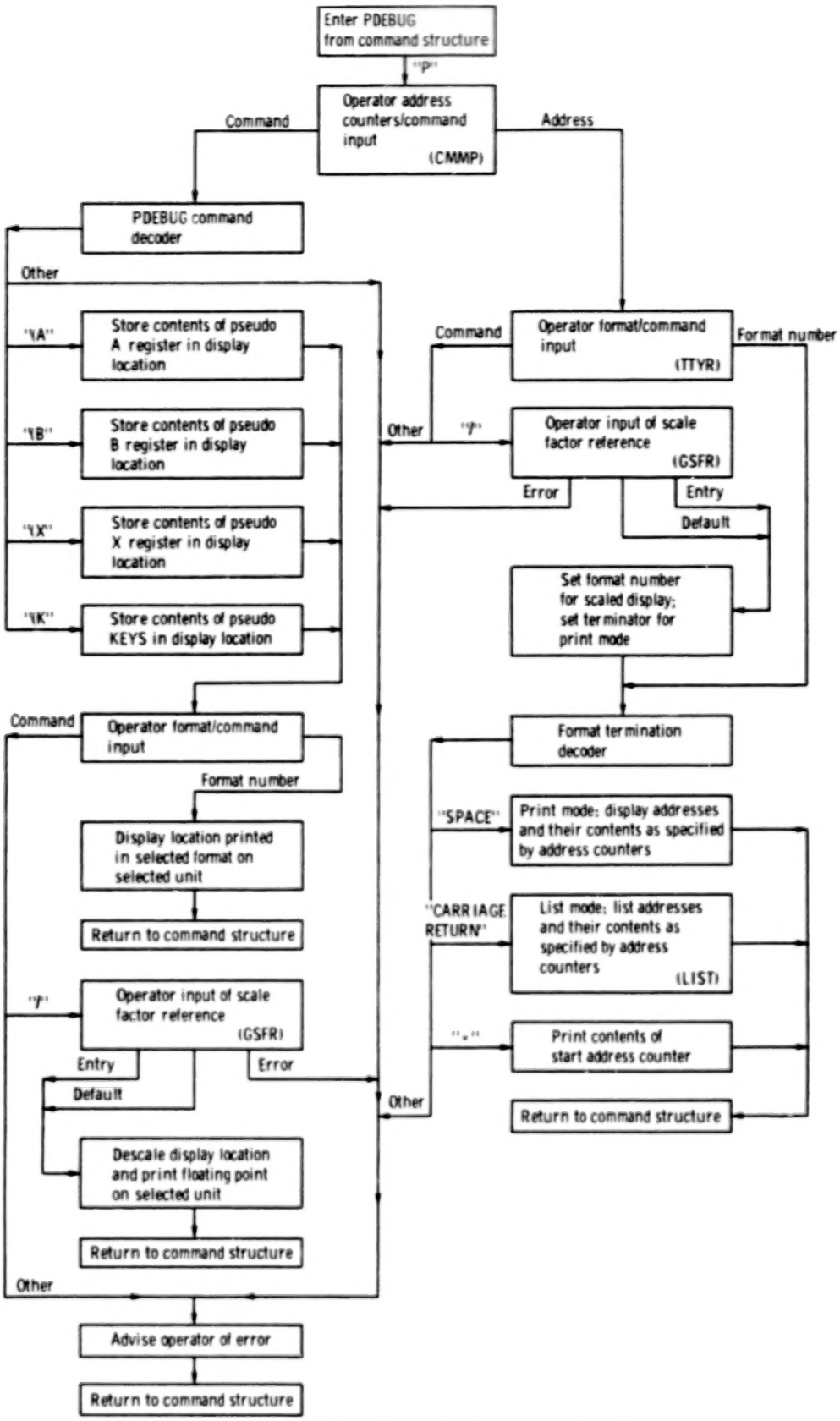


Figure 21. - Functional diagram of display command (PDEBUG).

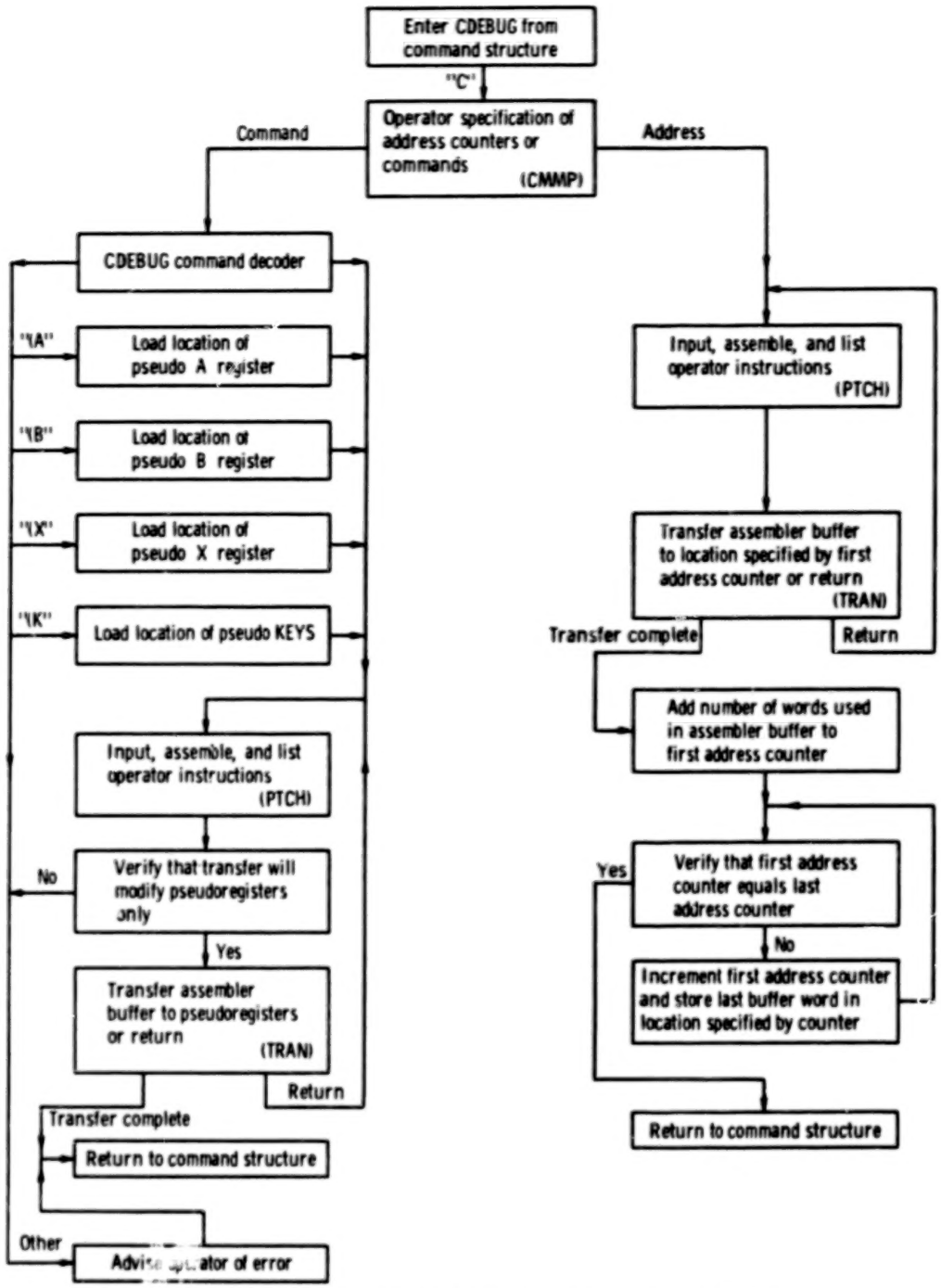

Enter CDEBUG from command structure
"C"
Operator specification of address counters or commands (CMMP)
Command
Address
CDEBUG command decoder
"A"
Load location of pseudo A register
"B"
Load location of pseudo B register
"X"
Load location of pseudo X register
"K"
Load location of pseudo KEYS
Input, assemble, and list operator instructions (PTCH)
No
Verify that transfer will modify pseudoregisters only
Yes
Transfer assembler buffer to pseudoregisters or return (TRAN)
Return
Transfer complete
Return to command structure
Other
Advise operator of error
Input, assemble, and list operator instructions (PTCH)
Transfer assembler buffer to location specified by first address counter or return (TRAN)
Transfer complete
Return
Add number of words used in assembler buffer to first address counter
Yes
Verify that first address counter equals last address counter
No
Increment first address counter and store last buffer word in location specified by counter
Return to command structure

Figure 22. - Functional diagram of register/memory change command (CDEBUG).

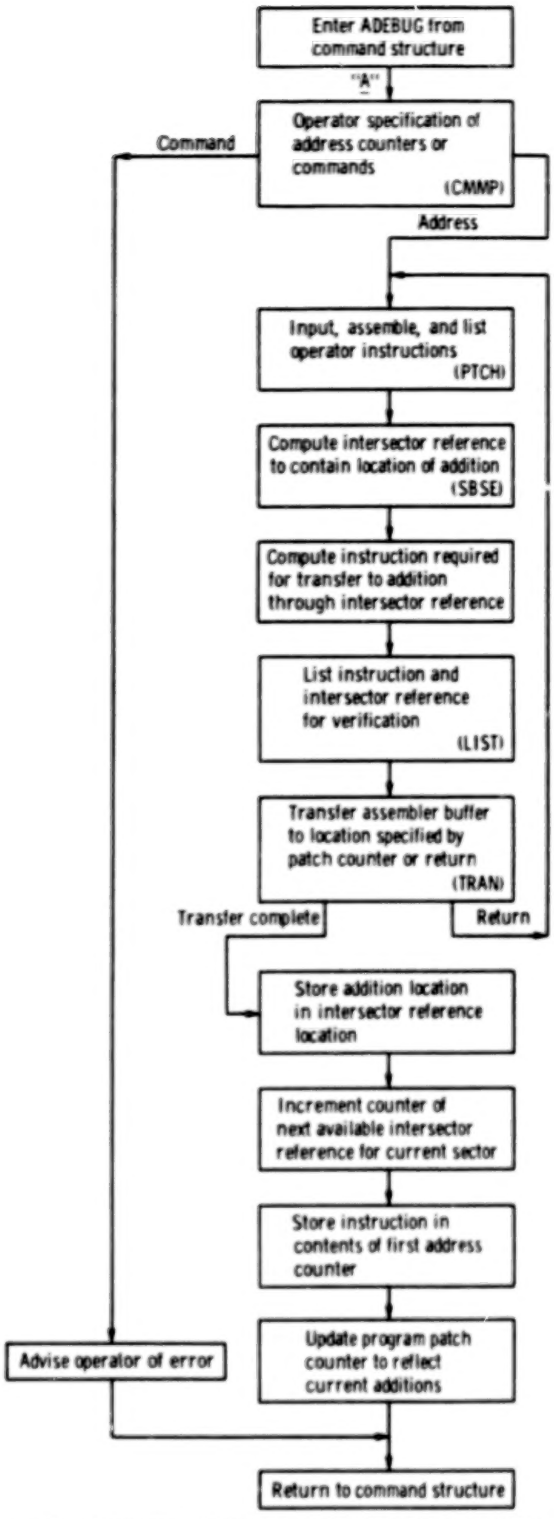

Enter ADEBUG from command structure
"A"
Operator specification of address counters or commands
(CMMP)
Command
Address
Input, assemble, and list operator instructions
(PTCH)
Compute intersector reference to contain location of addition
(SBSE)
Compute instruction required for transfer to addition through intersector reference
List instruction and intersector reference for verification
(LIST)
Transfer assembler buffer to location specified by patch counter or return
(TRAN)
Transfer complete
Return
Store addition location in intersector reference location
Increment counter of next available intersector reference for current sector
Store instruction in contents of first address counter
Update program patch counter to reflect current additions
Advise operator of error
Return to command structure

Figure 23. - Functional diagram of program addition command (ADEBUG).

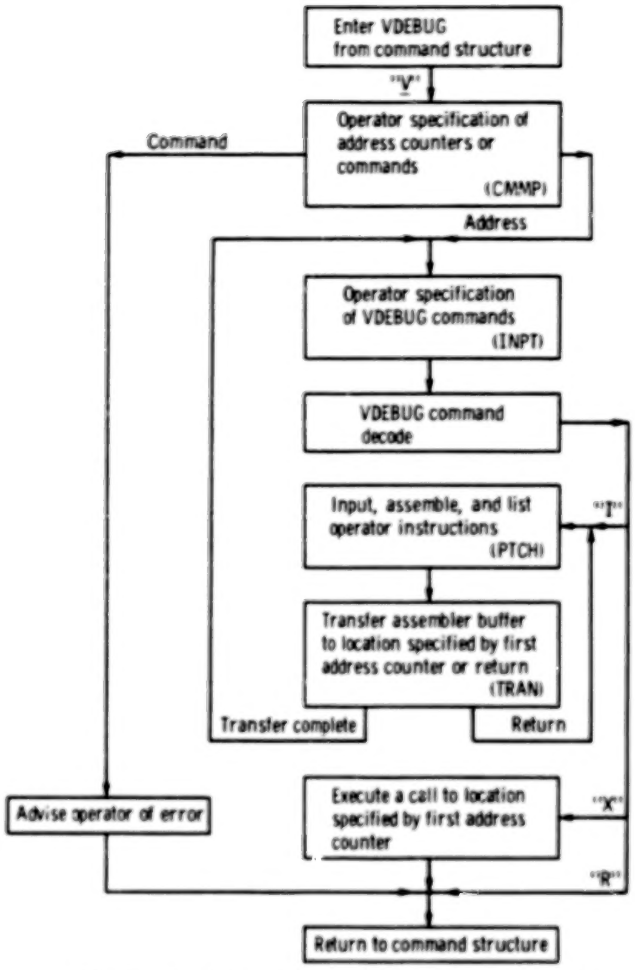
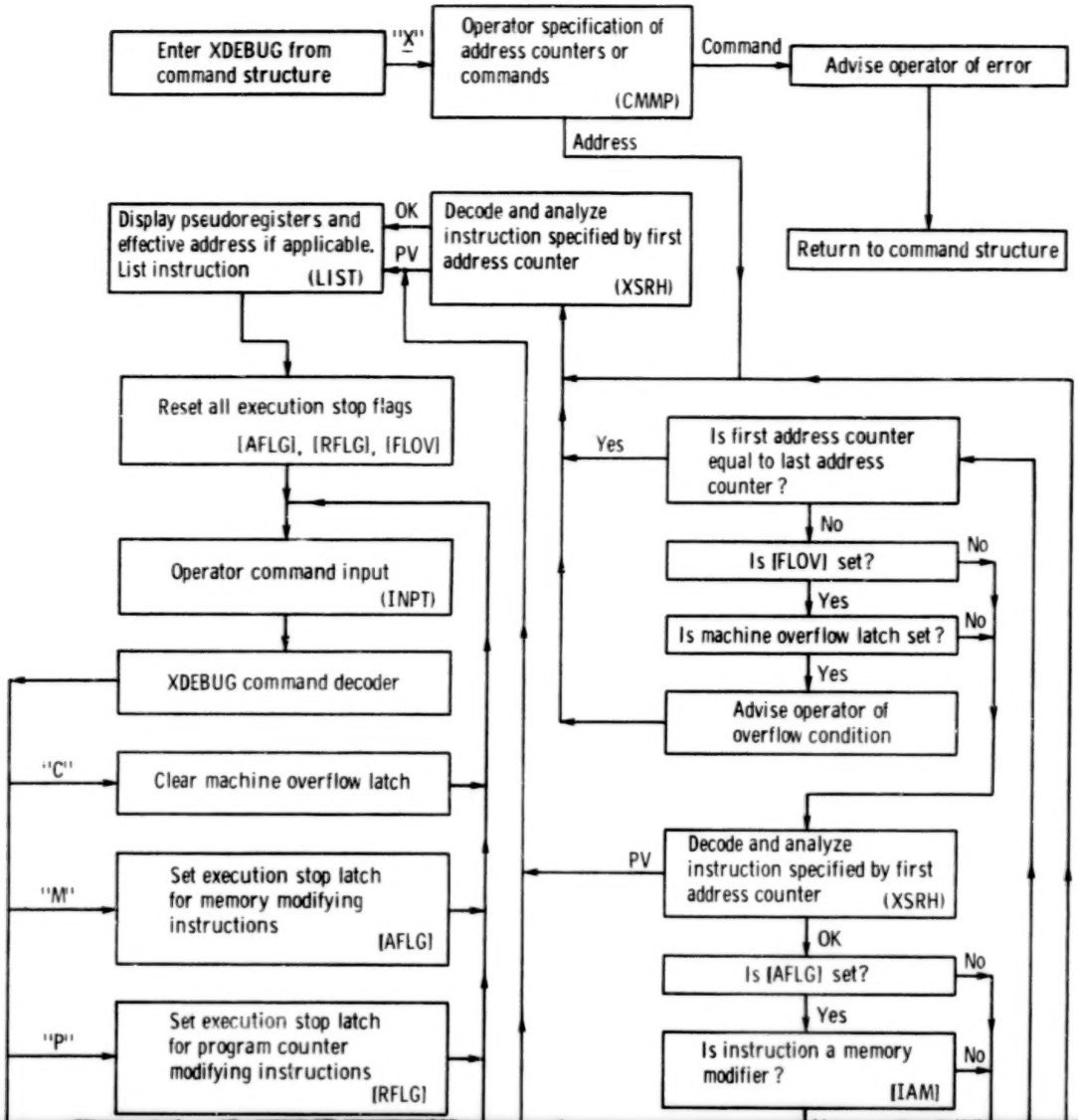


Figure 24. - Functional diagram of subroutine specification and execution command (VDEBUG).


Enter XDEBUG from command structure
"X"
Operator specification of address counters or commands (CMMP)
Command
Advise operator of error
Address
Return to command structure
Display pseudoregisters and effective address if applicable. List instruction (LIST)
OK
PV
Decode and analyze instruction specified by first address counter (XSRH)
Reset all execution stop flags [AFLG], [RFLG], [FLOV]
Yes
Is first address counter equal to last address counter ?
No
Is [FLOV] set?
No
Yes
Is machine overflow latch set ?
No
Yes
Advise operator of overflow condition
Operator command input (INPT)
XDEBUG command decoder
"C"
Clear machine overflow latch
PV
Decode and analyze instruction specified by first address counter (XSRH)
"M"
Set execution stop latch for memory modifying instructions [AFLG]
OK
Is [AFLG] set?
No
Yes
"P"
Set execution stop latch for program counter modifying instructions [RFLG]
Is instruction a memory modifier ?
No
[IAM]

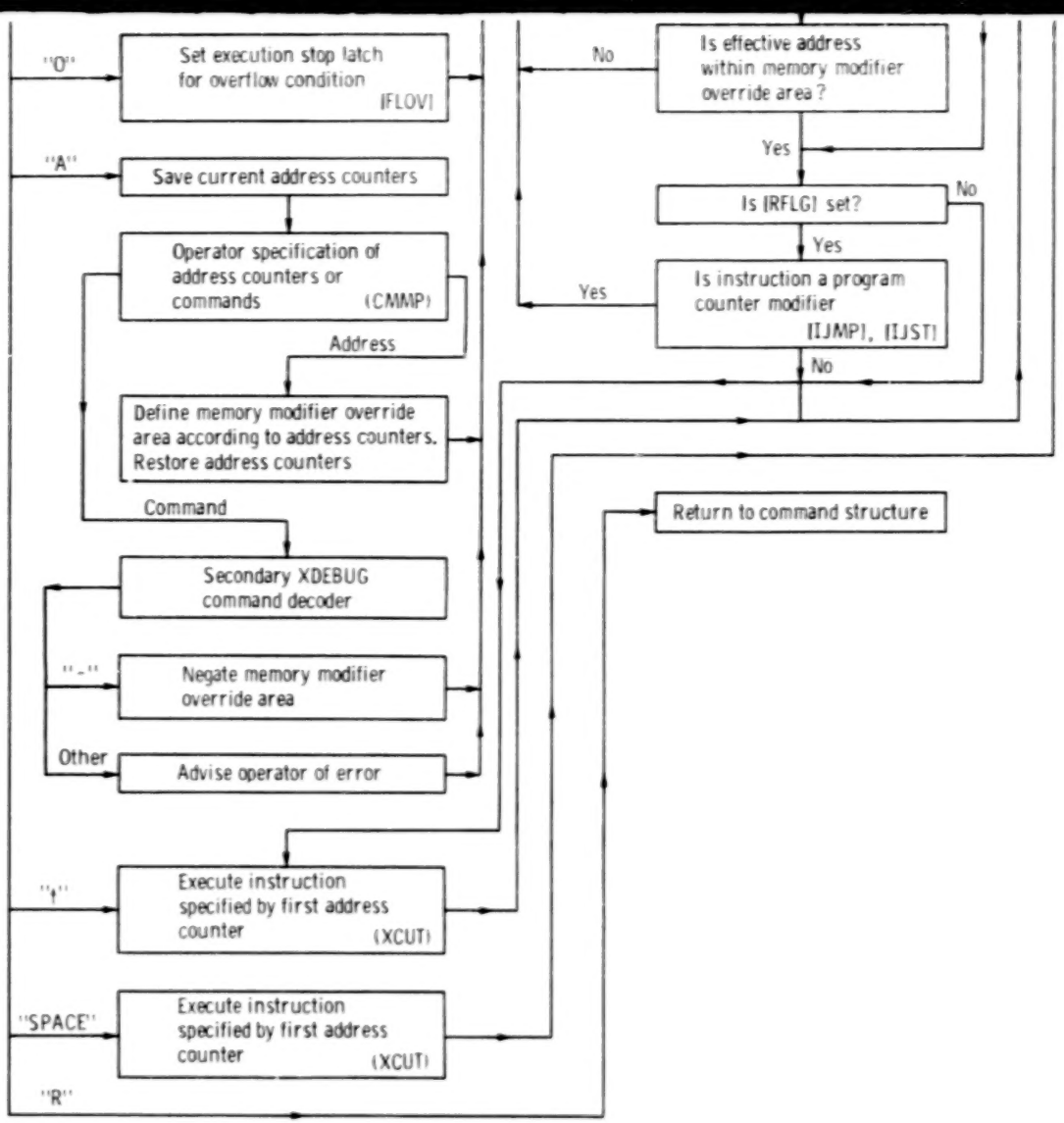


Figure 25. - Functional diagram of execution command (XDEBUG).

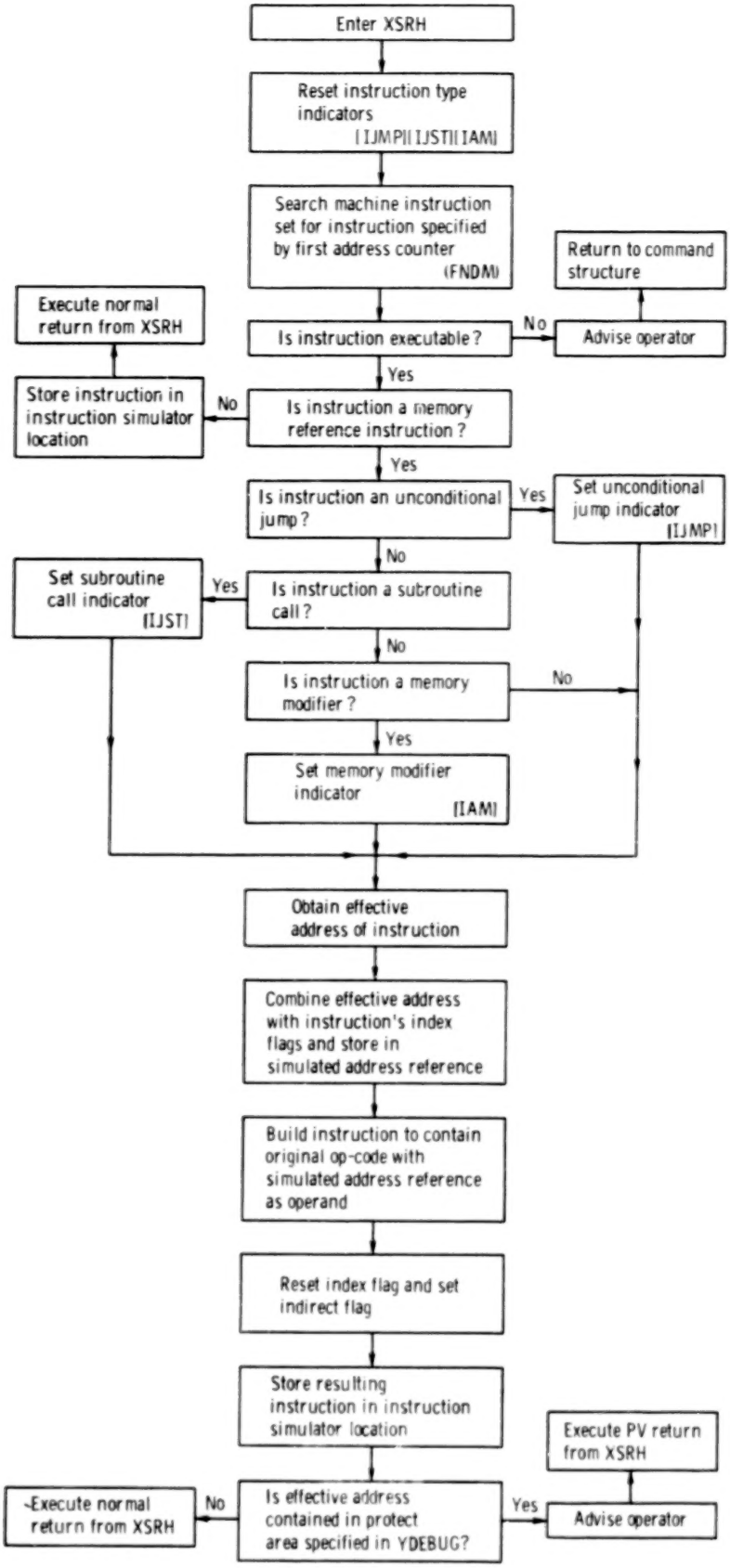

Enter XSRH
Reset instruction type indicators [IJMP][IJST][IAM]
Search machine instruction set for instruction specified by first address counter (FNDM)
Return to command structure
Execute normal return from XSRH
Is instruction executable?
No
Advise operator
Yes
Store instruction in instruction simulator location
No
Is instruction a memory reference instruction?
Yes
Is instruction an unconditional jump?
Yes
Set unconditional jump indicator [IJMP]
No
Set subroutine call indicator [IJST]
Yes
Is instruction a subroutine call?
No
Is instruction a memory modifier?
No
Yes
Set memory modifier indicator [IAM]
Obtain effective address of instruction
Combine effective address with instruction's index flags and store in simulated address reference
Build instruction to contain original op-code with simulated address reference as operand
Reset index flag and set indirect flag
Store resulting instruction in instruction simulator location
Execute PV return from XSRH
Execute normal return from XSRH
No
Is effective address contained in protect area specified in YDEBUG?
Yes
Advise operator

Figure 25. - Continued.

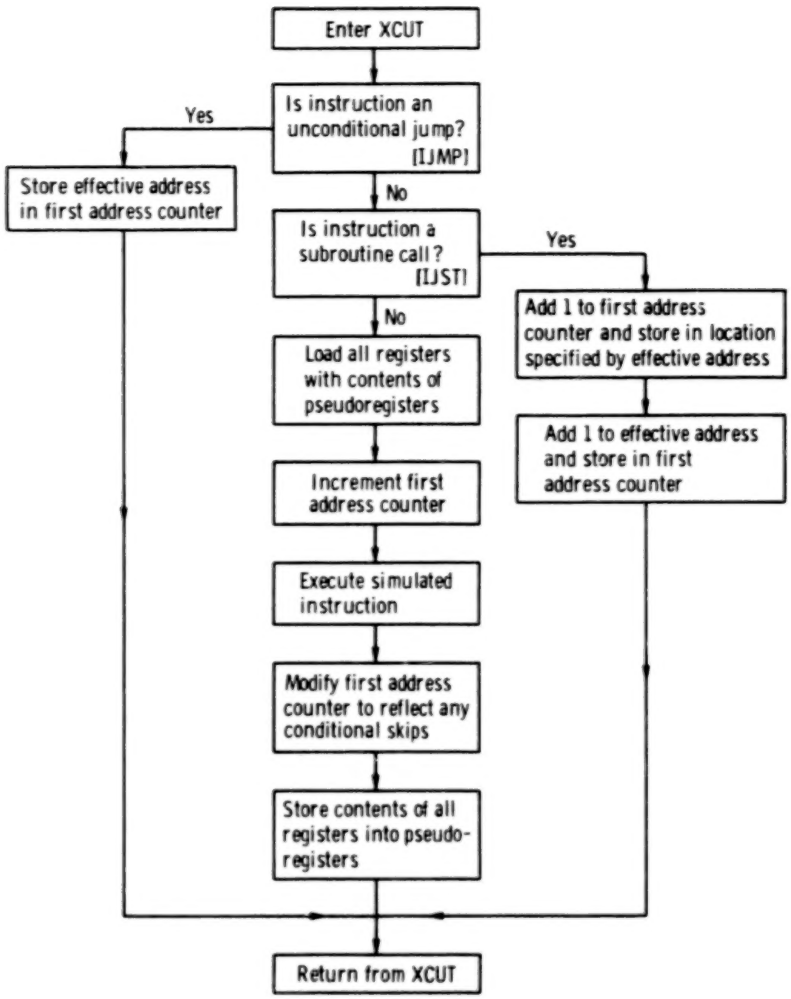

Enter XCUT
Is instruction an unconditional jump? [IJMP]
Yes
Store effective address in first address counter
No
Is instruction a subroutine call? [IJST]
Yes
Add 1 to first address counter and store in location specified by effective address
Add 1 to effective address and store in first address counter
No
Load all registers with contents of pseudoregisters
Increment first address counter
Execute simulated instruction
Modify first address counter to reflect any conditional skips
Store contents of all registers into pseudo-registers
Return from XCUT

Figure 25. - Concluded.

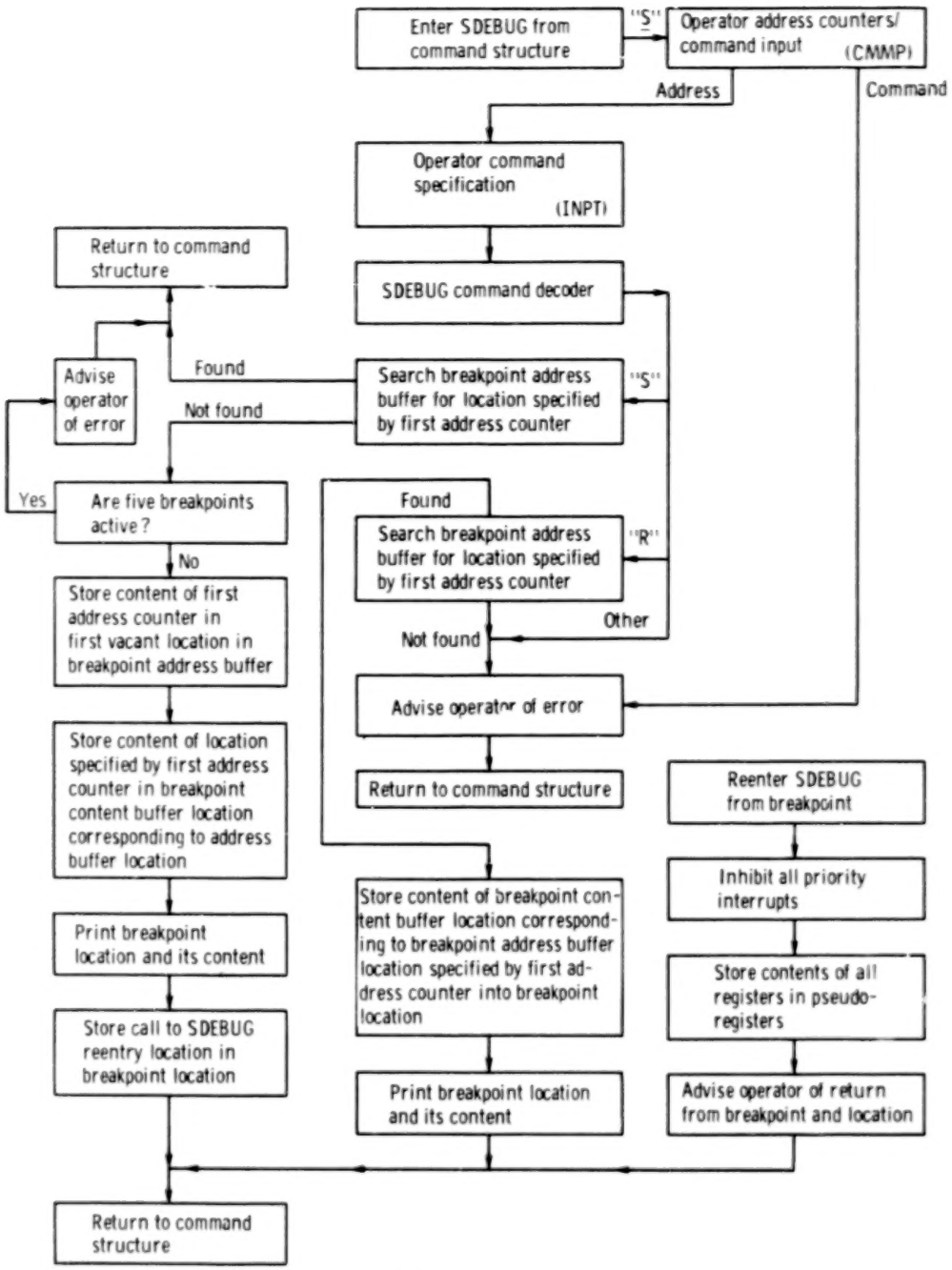

Enter SDEBUG from command structure
"S"
Operator address counters/ command input (CMMP)
Address
Command
Operator command specification (INPT)
SDEBUG command decoder
Return to command structure
Advise operator of error
Found
Not found
Search breakpoint address buffer for location specified by first address counter
"S"
Yes
Are five breakpoints active?
No
Store content of first address counter in first vacant location in breakpoint address buffer
Store content of location specified by first address counter in breakpoint content buffer location corresponding to address buffer location
Print breakpoint location and its content
Store call to SDEBUG reentry location in breakpoint location
Found
Search breakpoint address buffer for location specified by first address counter
"R"
Other
Not found
Advise operator of error
Return to command structure
Store content of breakpoint content buffer location corresponding to breakpoint address buffer location specified by first address counter into breakpoint location
Print breakpoint location and its content
Reenter SDEBUG from breakpoint
Inhibit all priority interrupts
Store contents of all registers in pseudo-registers
Advise operator of return from breakpoint and location
Return to command structure

Figure 26. - Functional diagram of breakpoint command (SDEBUG).

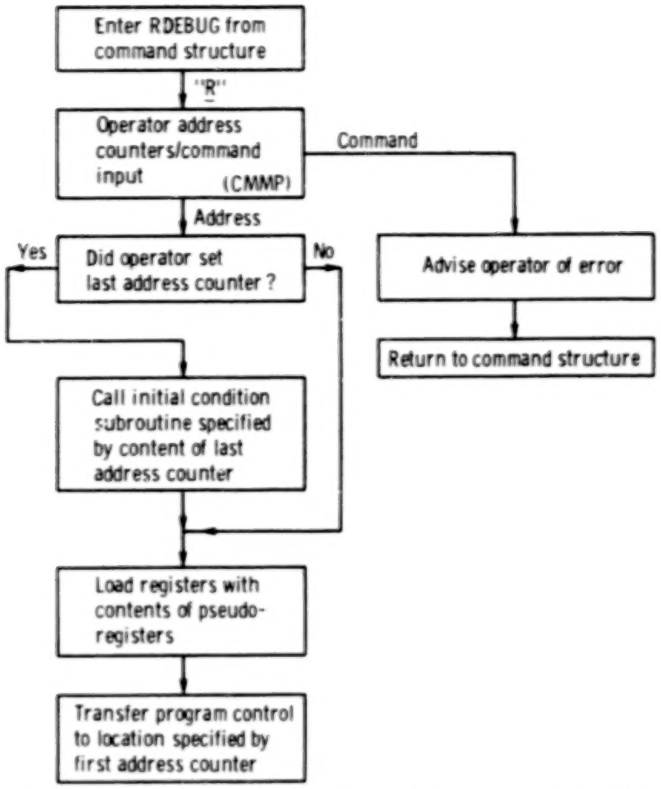

Enter RDEBUG from command structure
"R"
Operator address counters/command input (CMMP)
Command
Address
Yes
Did operator set last address counter ?
No
Advise operator of error
Return to command structure
Call initial condition subroutine specified by content of last address counter
Load registers with contents of pseudo-registers
Transfer program control to location specified by first address counter

Figure 27. - Functional diagram of program return instruction (RDEBUG).

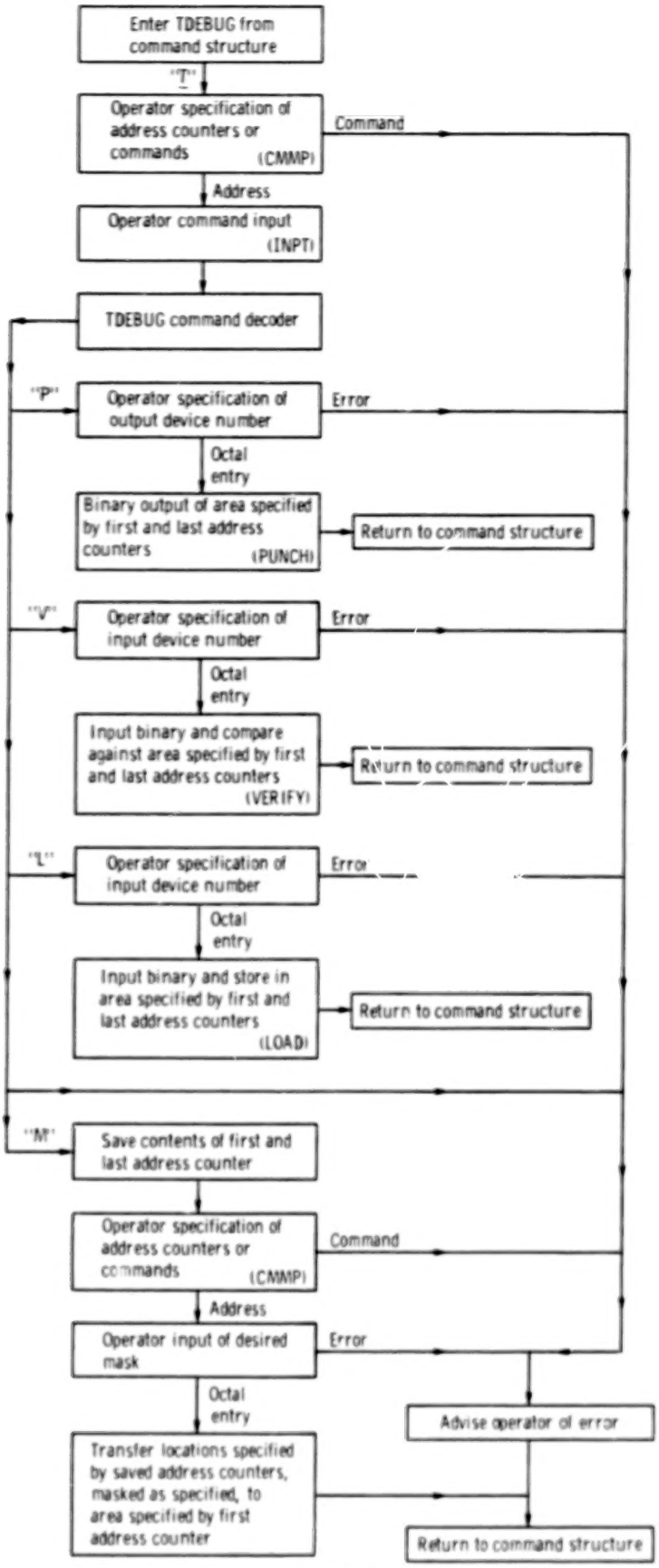
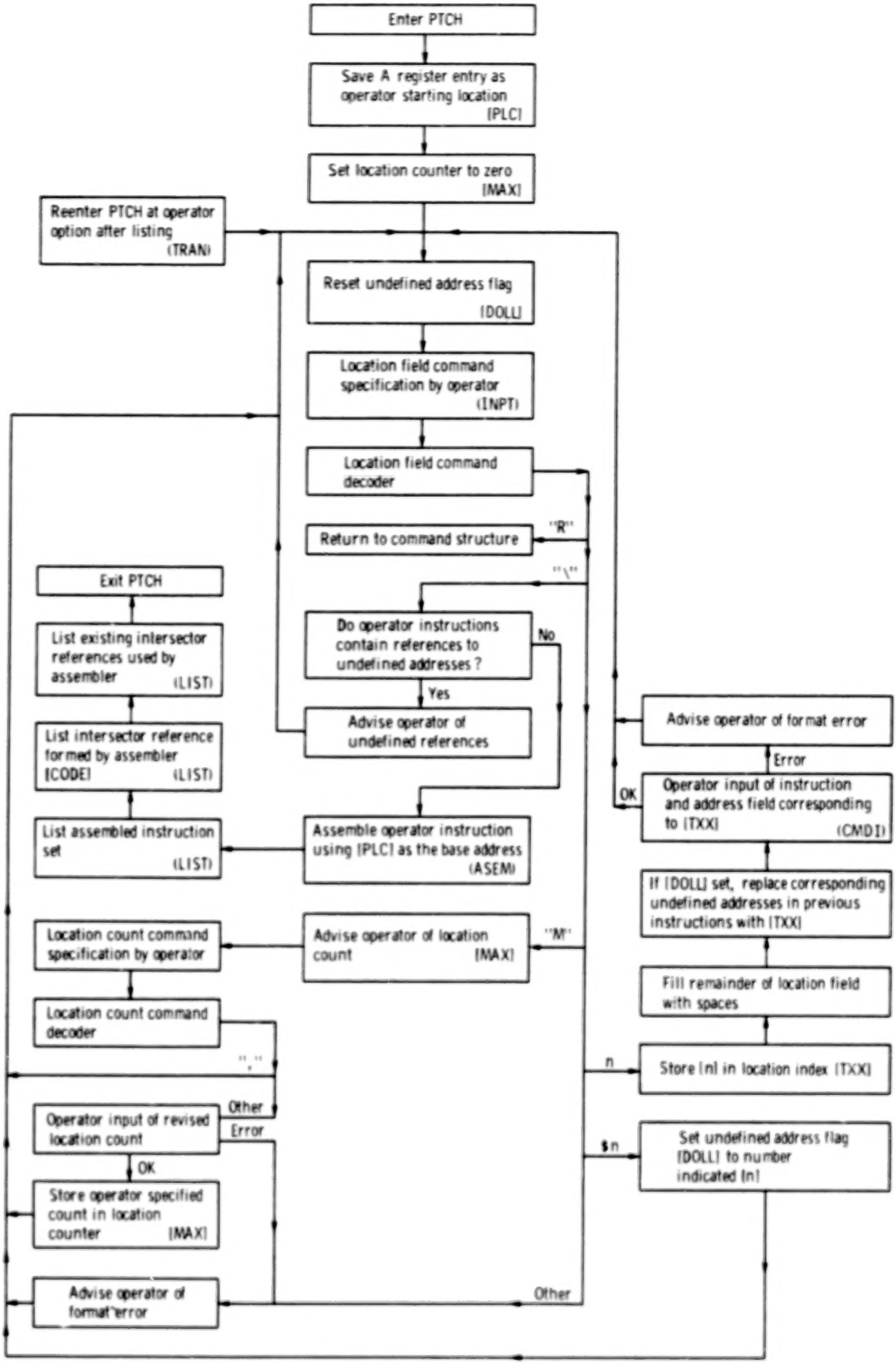
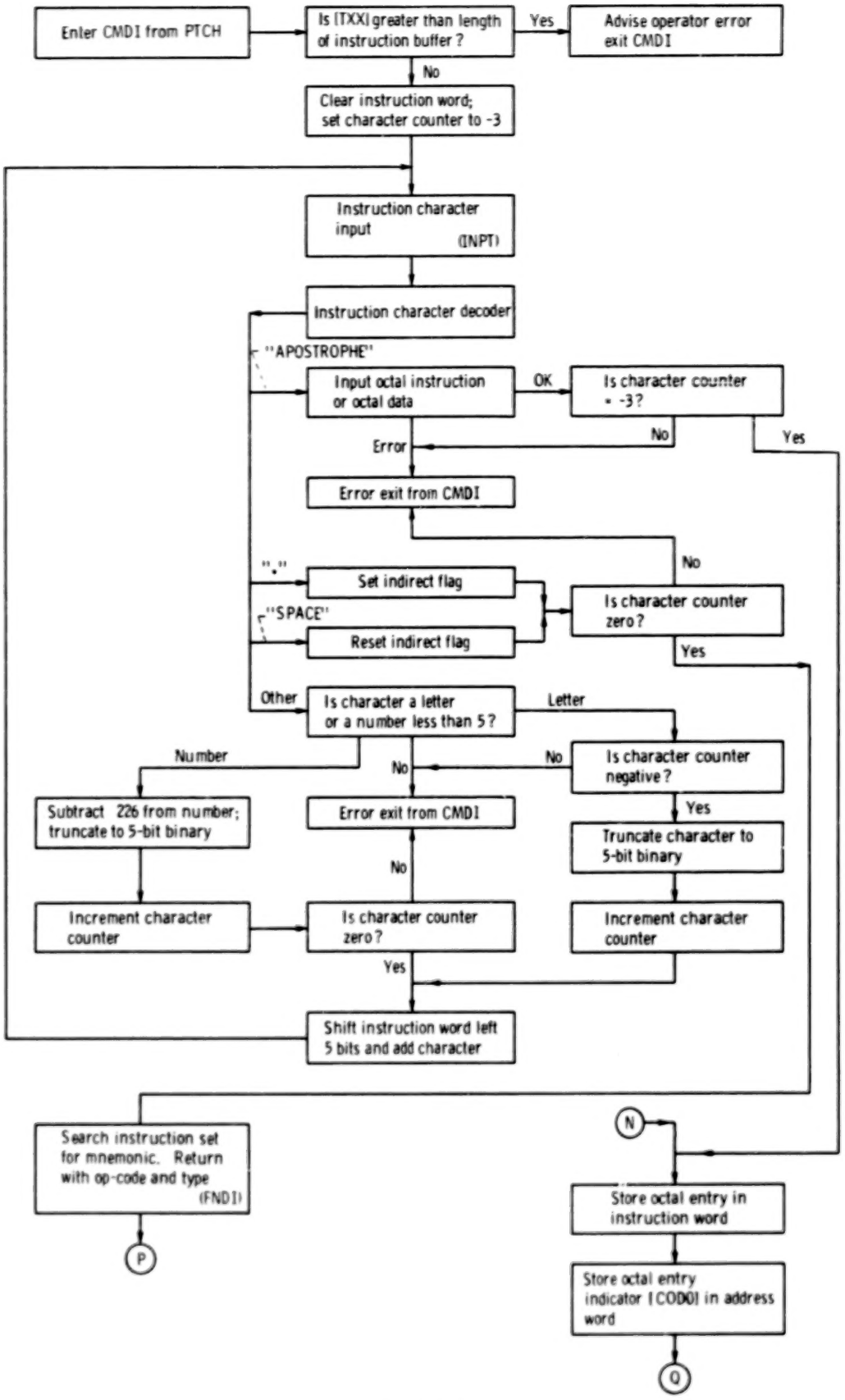

Enter TDEBUG from command structure
"T"
Operator specification of address counters or commands (CMMP)
Command
Address
Operator command input (INPT)
TDEBUG command decoder
"P"
Operator specification of output device number
Error
Octal entry
Binary output of area specified by first and last address counters (PUNCH)
Return to command structure
"V"
Operator specification of input device number
Error
Octal entry
Input binary and compare against area specified by first and last address counters (VERIFY)
Return to command structure
"L"
Operator specification of input device number
Error
Octal entry
Input binary and store in area specified by first and last address counters (LOAD)
Return to command structure
"M"
Save contents of first and last address counter
Operator specification of address counters or commands (CMMP)
Command
Address
Operator input of desired mask
Error
Octal entry
Advise operator of error
Transfer locations specified by saved address counters, masked as specified, to area specified by first address counter
Return to command structure

Figure 28. - Functional diagram of data transfer command (TDEBUG).


Enter PTCH
Save A register entry as operator starting location [PLC]
Set location counter to zero [MAX]
Reenter PTCH at operator option after listing (TRAN)
Reset undefined address flag [DOLL]
Location field command specification by operator (INPT)
Location field command decoder
"R"
Return to command structure
"\"
Exit PTCH
Do operator instructions contain references to undefined addresses ?
No
Yes
List existing intersector references used by assembler (LIST)
Advise operator of undefined references
Advise operator of format error
Error
List intersector reference formed by assembler [CODE] (LIST)
OK
Operator input of instruction and address field corresponding to [TXX] (CMDI)
List assembled instruction set (LIST)
Assemble operator instruction using [PLC] as the base address (ASEM)
If [DOLL] set, replace corresponding undefined addresses in previous instructions with [TXX]
Location count command specification by operator
Advise operator of location count [MAX]
"M"
Fill remainder of location field with spaces
Location count command decoder
n
Store [n] in location index [TXX]
Other
Operator input of revised location count
Error
$n
Set undefined address flag [DOLL] to number indicated [n]
OK
Store operator specified count in location counter [MAX]
Advise operator of format error
Other

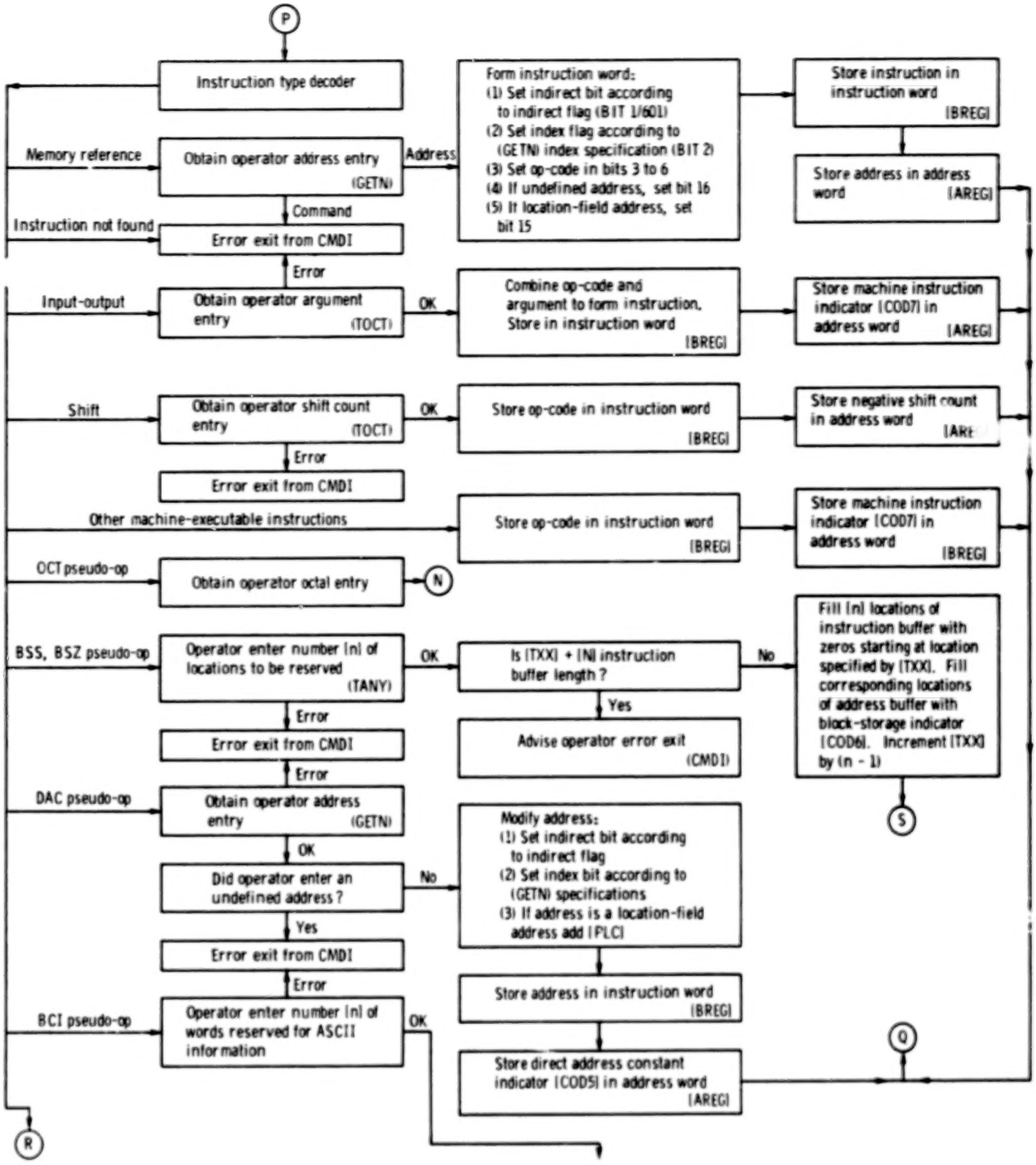
(a) Part I (PTCH).

Figure 29. - Functional diagram of debug assembler.



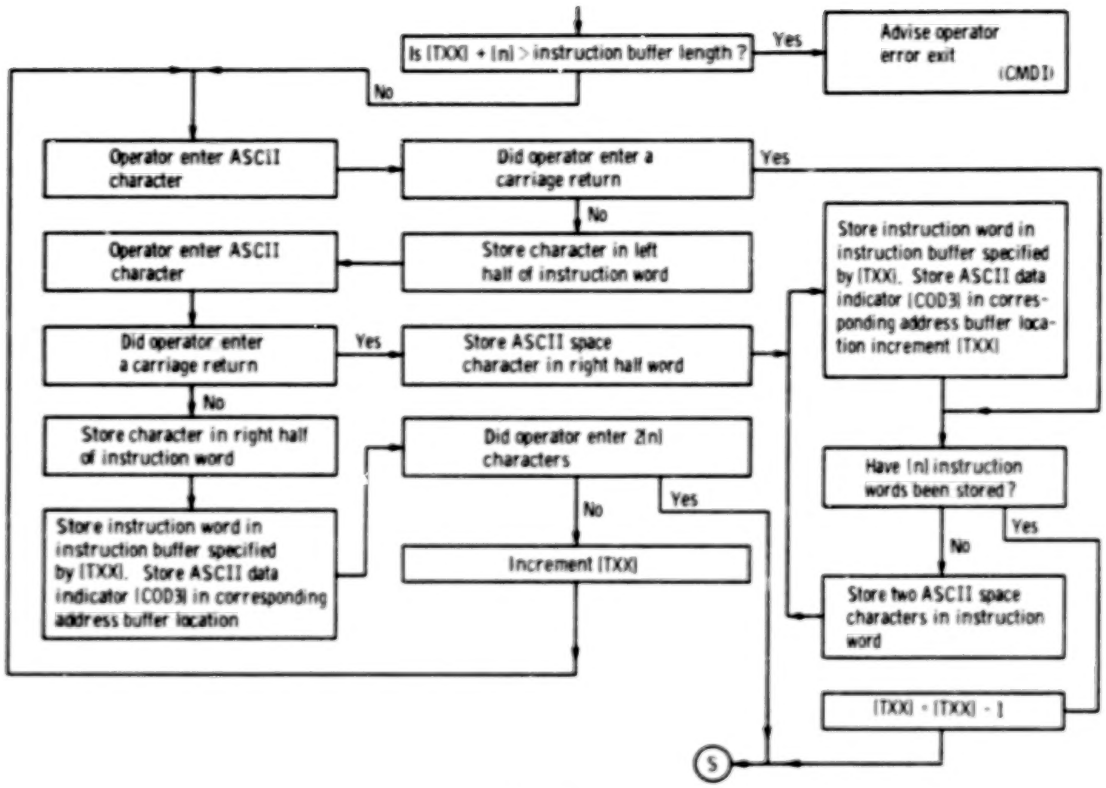
(b) Part II (CMDI).

Figure 29. - Continued.


P
Instruction type decoder
Memory reference
Obtain operator address entry (GETN)
Address
Form instruction word:
(1) Set indirect bit according to indirect flag (BIT 1/60)
(2) Set index flag according to (GETN) index specification (BIT 2)
(3) Set op-code in bits 3 to 6
(4) If undefined address, set bit 16
(5) If location-field address, set bit 15
Store instruction in instruction word [BREG]
Store address in address word [AREG]
Command
Instruction not found
Error exit from CMDI
Error
Input-output
Obtain operator argument entry (TOCT)
OK
Combine op-code and argument to form instruction. Store in instruction word [BREG]
Store machine instruction indicator [COD7] in address word [AREG]
Shift
Obtain operator shift count entry (TOCT)
OK
Store op-code in instruction word [BREG]
Store negative shift count in address word [AR
Error
Error exit from CMDI
Other machine-executable instructions
Store op-code in instruction word [BREG]
Store machine instruction indicator [COD7] in address word [BREG]
OCT pseudo-op
Obtain operator octal entry
N
BSS, BSZ pseudo-op
Operator enter number [n] of locations to be reserved (TANY)
OK
Is [TXX] + [N] instruction buffer length?
No
Yes
Fill [n] locations of instruction buffer with zeros starting at location specified by [TXX]. Fill corresponding locations of address buffer with block-storage indicator [COD6]. Increment [TXX] by (n - 1)
Error
Error exit from CMDI
Advise operator error exit (CMDI)
S
Error
DAC pseudo-op
Obtain operator address entry (GETN)
OK
Did operator enter an undefined address?
No
Modify address:
(1) Set indirect bit according to indirect flag
(2) Set index bit according to (GETN) specifications
(3) If address is a location-field address add [PLC]
Yes
Error exit from CMDI
Error
Store address in instruction word [BREG]
BCI pseudo-op
Operator enter number [n] of words reserved for ASCII information
OK
Q
Store direct address constant indicator [COD5] in address word [AREG]
R

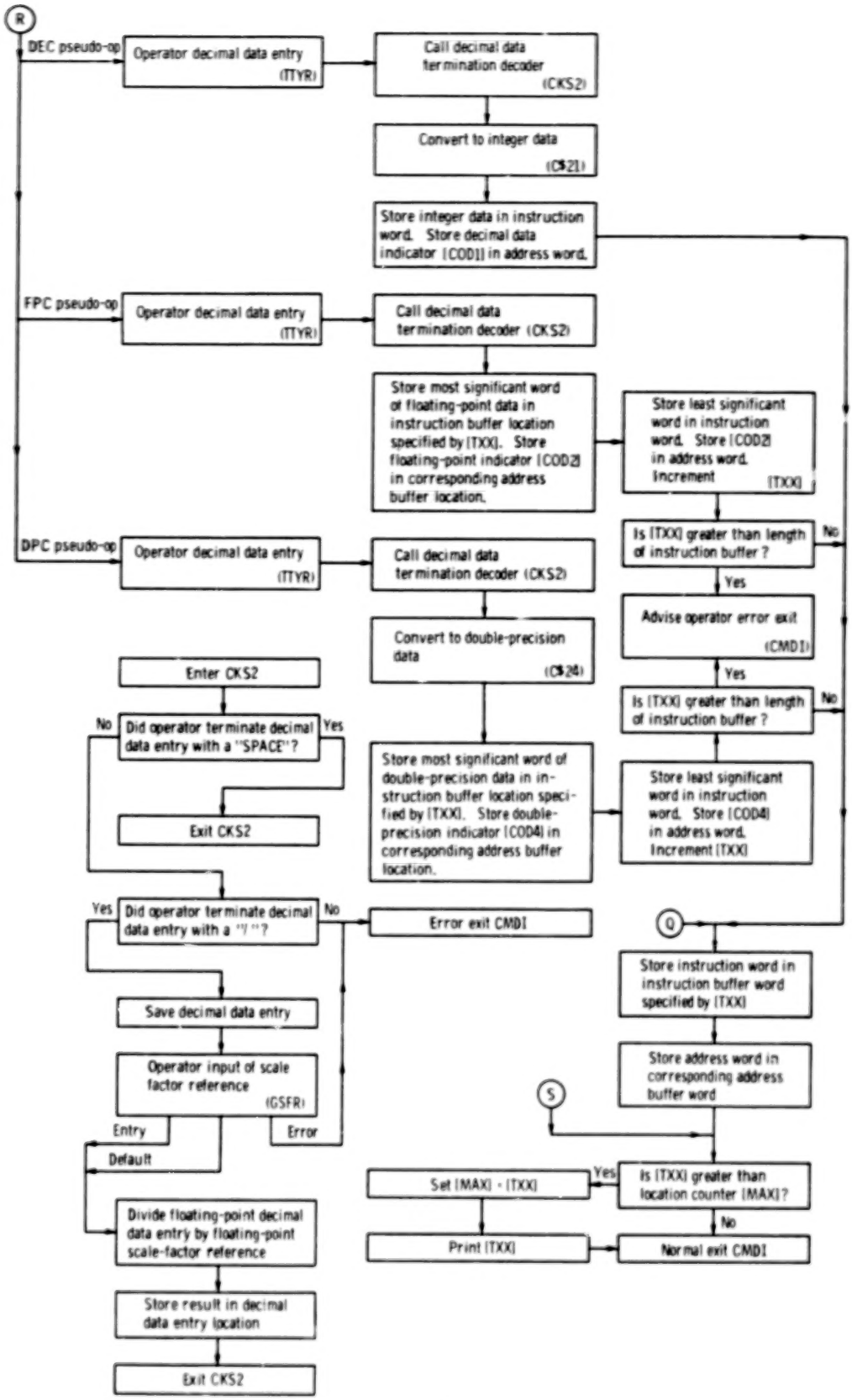
(b) Continued.

Figure 29. - Continued.


Is (TXX) + (n) > instruction buffer length ?
Yes
Advise operator error exit (CMD1)
No
Operator enter ASCII character
Did operator enter a carriage return
Yes
No
Store instruction word in instruction buffer specified by (TXX). Store ASCII data indicator (COD3) in corresponding address buffer location increment (TXX)
Operator enter ASCII character
Store character in left half of instruction word
Did operator enter a carriage return
Yes
Store ASCII space character in right half word
No
Store character in right half of instruction word
Did operator enter 2(n) characters
Have (n) instruction words been stored?
Yes
No
Yes
No
Store instruction word in instruction buffer specified by (TXX). Store ASCII data indicator (COD3) in corresponding address buffer location
Increment (TXX)
Store two ASCII space characters in instruction word
(TXX) = (TXX) - 1
S

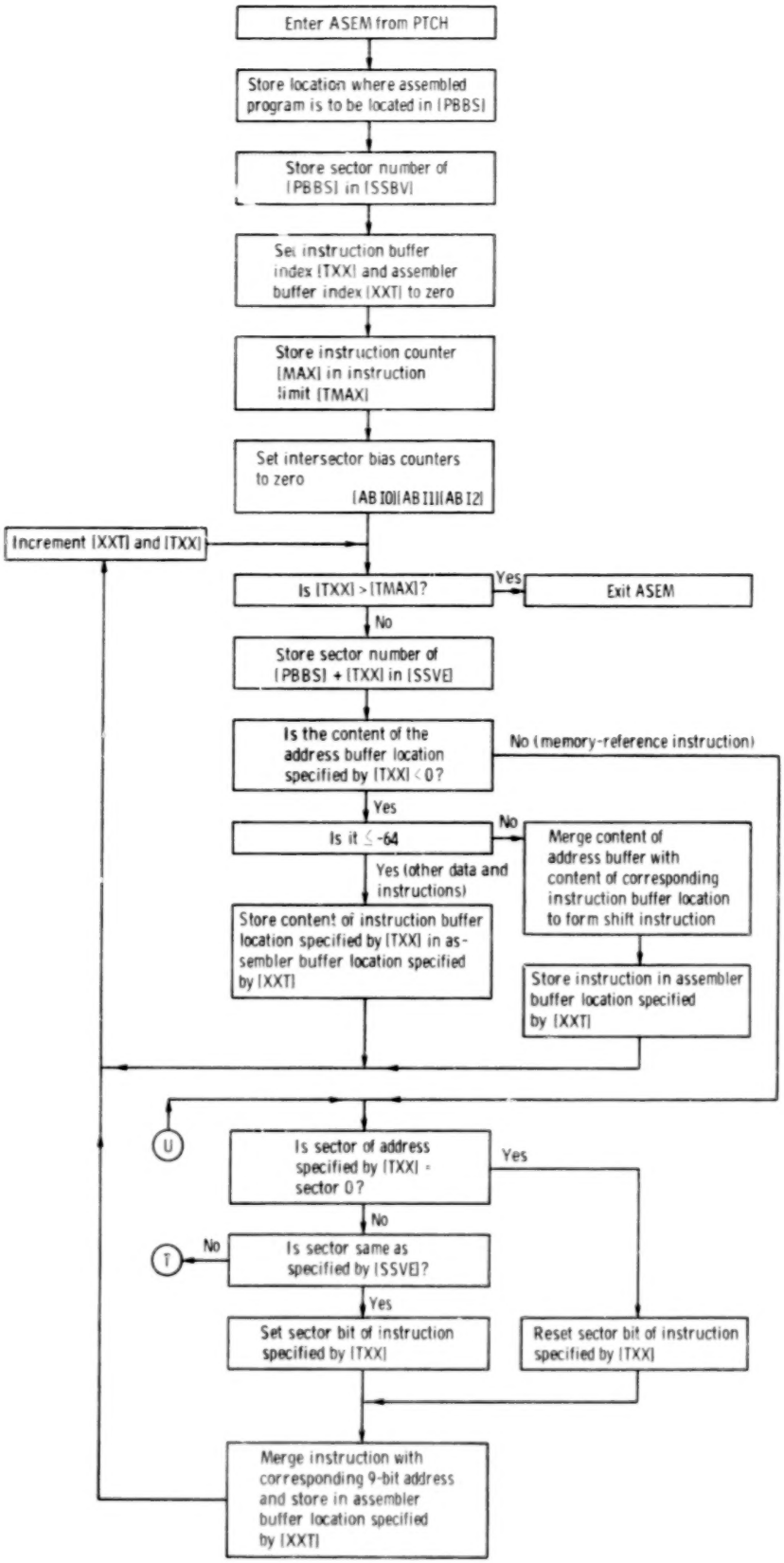
(b) Continued.

Figure 29. - Continued.


R
DEC pseudo-op
Operator decimal data entry (TTYR)
Call decimal data termination decoder (CKS2)
Convert to integer data (CS21)
Store integer data in instruction word. Store decimal data indicator [COD1] in address word.
FPC pseudo-op
Operator decimal data entry (TTYR)
Call decimal data termination decoder (CKS2)
Store most significant word of floating-point data in instruction buffer location specified by [TXX]. Store floating-point indicator [COD2] in corresponding address buffer location.
Store least significant word in instruction word. Store [COD2] in address word. Increment [TXX]
Is [TXX] greater than length of instruction buffer ?
No
Yes
Advise operator error exit (CMDI)
DPC pseudo-op
Operator decimal data entry (TTYR)
Call decimal data termination decoder (CKS2)
Convert to double-precision data (CS24)
Enter CKS2
No
Did operator terminate decimal data entry with a "SPACE"?
Yes
Exit CKS2
Store most significant word of double-precision data in instruction buffer location specified by [TXX]. Store double-precision indicator [COD4] in corresponding address buffer location.
Is [TXX] greater than length of instruction buffer ?
No
Yes
Store least significant word in instruction word. Store [COD4] in address word. Increment [TXX]
Yes
Did operator terminate decimal data entry with a "/"?
No
Error exit CMDI
Q
Store instruction word in instruction buffer word specified by [TXX]
Store address word in corresponding address buffer word
S
Save decimal data entry
Operator input of scale factor reference (GSFR)
Entry
Error
Default
Set [MAX] = [TXX]
Yes
Is [TXX] greater than location counter [MAX]?
No
Divide floating-point decimal data entry by floating-point scale-factor reference
Print [TXX]
Normal exit CMDI
Store result in decimal data entry location
Exit CKS2

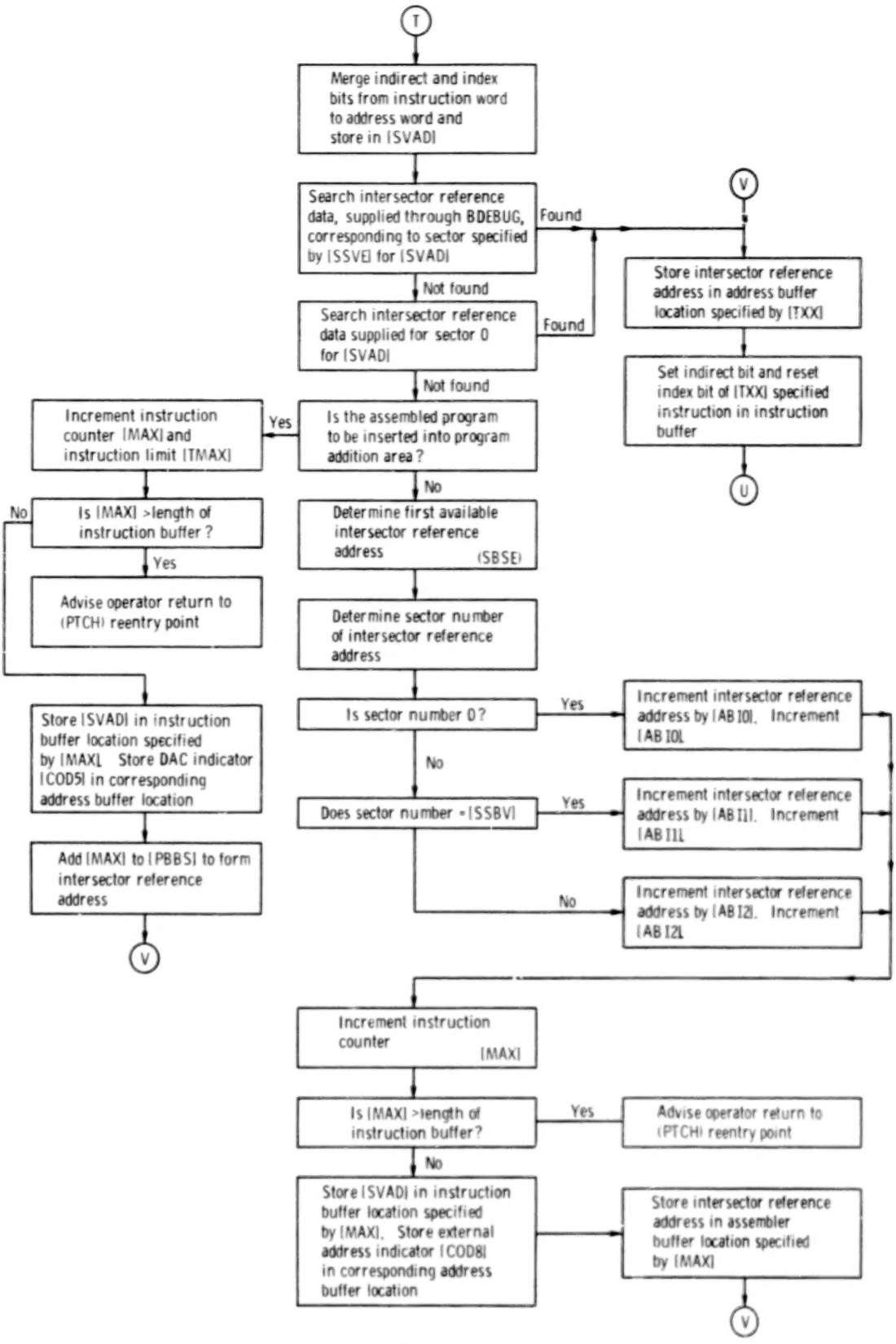
(b) Concluded.

Figure 29. - Continued.


Enter ASEM from PTCH
Store location where assembled program is to be located in [PBBS]
Store sector number of [PBBS] in [SSBV]
Set instruction buffer index [TXX] and assembler buffer index [XXT] to zero
Store instruction counter [MAX] in instruction limit [TMAX]
Set intersector bias counters to zero [AB I0][AB I1][AB I2]
Increment [XXT] and [TXX]
Is [TXX] > [TMAX]?
Yes
Exit ASEM
No
Store sector number of [PBBS] + [TXX] in [SSVE]
Is the content of the address buffer location specified by [TXX] < 0?
No (memory-reference instruction)
Yes
Is it ≤ -64
No
Merge content of address buffer with content of corresponding instruction buffer location to form shift instruction
Yes (other data and instructions)
Store content of instruction buffer location specified by [TXX] in assembler buffer location specified by [XXT]
Store instruction in assembler buffer location specified by [XXT]
U
Is sector of address specified by [TXX] = sector 0?
Yes
No
T
No
Is sector same as specified by [SSVE]?
Yes
Set sector bit of instruction specified by [TXX]
Reset sector bit of instruction specified by [TXX]
Merge instruction with corresponding 9-bit address and store in assembler buffer location specified by [XXT]

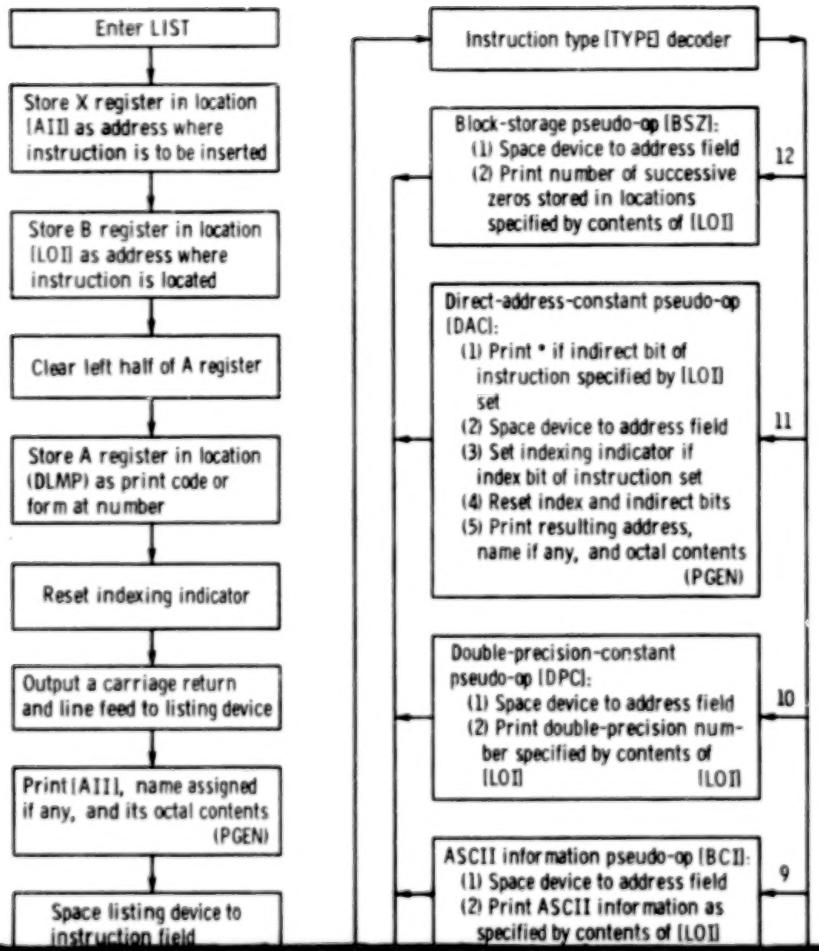
(c) Part III (ASEM).

Figure 29. - Continued.


T
Merge indirect and index bits from instruction word to address word and store in [SVAD]
Search intersector reference data, supplied through BDEBUG, corresponding to sector specified by [SSVE] for [SVAD]
Found
V
Store intersector reference address in address buffer location specified by [TXX]
Set indirect bit and reset index bit of [TXX] specified instruction in instruction buffer
U
Not found
Search intersector reference data supplied for sector 0 for [SVAD]
Found
Not found
Increment instruction counter [MAX] and instruction limit [TMAX]
Yes
Is the assembled program to be inserted into program addition area?
No
Determine first available intersector reference address (SBSE)
No
Is [MAX] >length of instruction buffer?
Yes
Advise operator return to (PTCH) reentry point
Determine sector number of intersector reference address
Store [SVAD] in instruction buffer location specified by [MAX]. Store DAC indicator [COD5] in corresponding address buffer location
Is sector number 0?
Yes
Increment intersector reference address by [AB10]. Increment [AB10L
No
Does sector number = [SSBV]
Yes
Increment intersector reference address by [AB11]. Increment [AB11L
No
Increment intersector reference address by [AB12]. Increment [AB12L
Add [MAX] to [PBBS] to form intersector reference address
V
Increment instruction counter [MAX]
Is [MAX] >length of instruction buffer?
Yes
Advise operator return to (PTCH) reentry point
No
Store [SVAD] in instruction buffer location specified by [MAX]. Store external address indicator [COD8] in corresponding address buffer location
Store intersector reference address in assembler buffer location specified by [MAX]
V

(c) Concluded.

Figure 29. - Concluded.


Enter LIST
Store X register in location [AII] as address where instruction is to be inserted
Store B register in location [LOI] as address where instruction is located
Clear left half of A register
Store A register in location (DLMP) as print code or form at number
Reset indexing indicator
Output a carriage return and line feed to listing device
Print [AII], name assigned if any, and its octal contents (PGEN)
Space listing device to instruction field
Instruction type [TYPE] decoder
Block-storage pseudo-op [BSZ]:
(1) Space device to address field
(2) Print number of successive zeros stored in locations specified by contents of [LOI]
12
Direct-address-constant pseudo-op [DAC]:
(1) Print * if indirect bit of instruction specified by [LOI] set
(2) Space device to address field
(3) Set indexing indicator if index bit of instruction set
(4) Reset index and indirect bits
(5) Print resulting address, name if any, and octal contents (PGEN)
11
Double-precision-constant pseudo-op [DPC]:
(1) Space device to address field
(2) Print double-precision number specified by contents of [LOI] [LOI]
10
ASCII information pseudo-op [BCI]:
(1) Space device to address field
(2) Print ASCII information as specified by contents of [LOI]
9

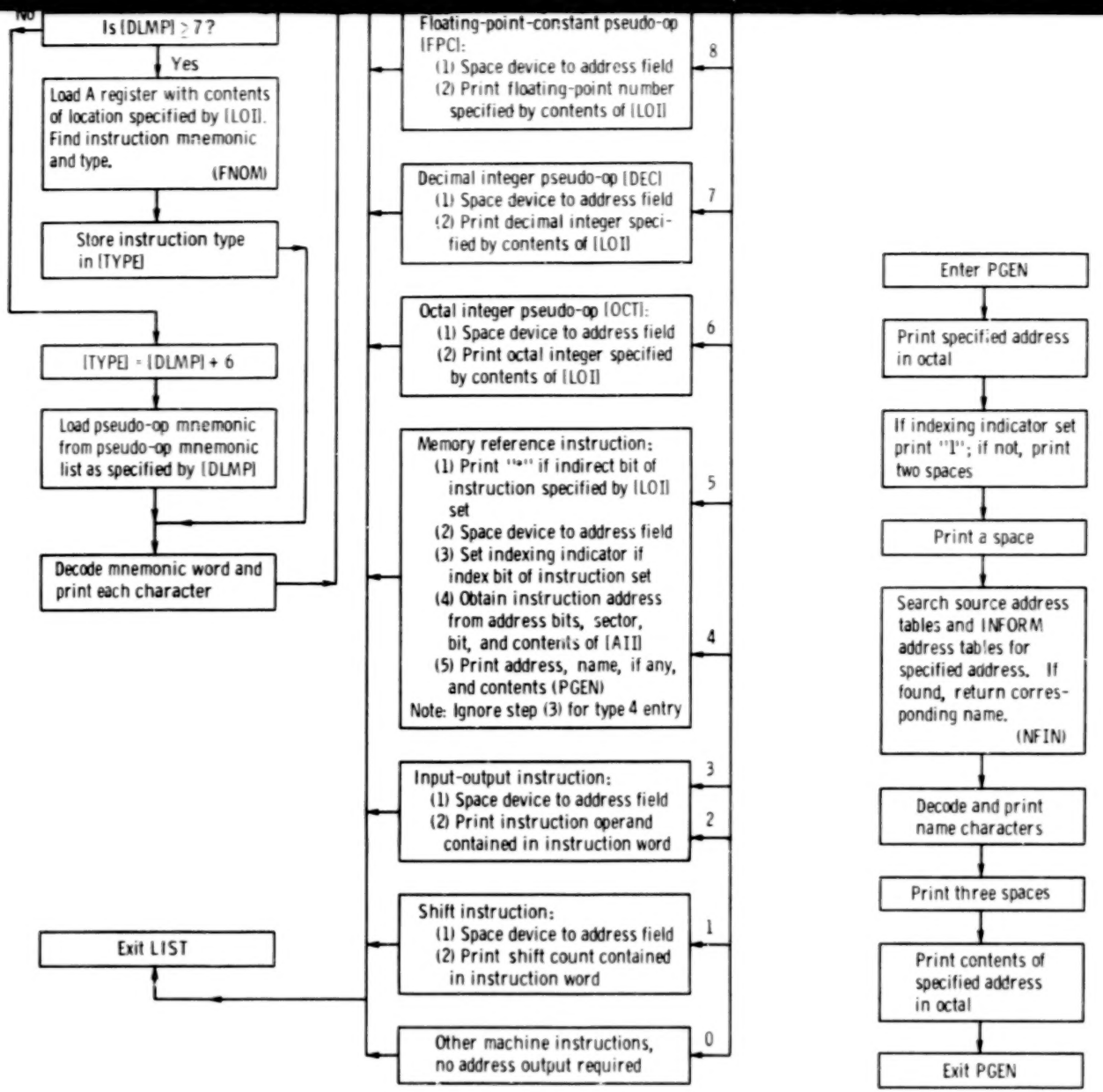

No
Is [DLMP] ≥ 7?
Yes
Load A register with contents of location specified by [LOI]. Find instruction mnemonic and type. (FNOM)
Store instruction type in [TYPE]
[TYPE] = [DLMP] + 6
Load pseudo-op mnemonic from pseudo-op mnemonic list as specified by [DLMP]
Decode mnemonic word and print each character
Exit LIST
Floating-point-constant pseudo-op [FPC]:
(1) Space device to address field
(2) Print floating-point number specified by contents of [LOI]
8
Decimal integer pseudo-op [DEC]
(1) Space device to address field
(2) Print decimal integer specified by contents of [LOI]
7
Octal integer pseudo-op [OCT]:
(1) Space device to address field
(2) Print octal integer specified by contents of [LOI]
6
Memory reference instruction:
(1) Print "*" if indirect bit of instruction specified by [LOI] set
(2) Space device to address field
(3) Set indexing indicator if index bit of instruction set
(4) Obtain instruction address from address bits, sector, bit, and contents of [AII]
(5) Print address, name, if any, and contents (PGEN)
Note: Ignore step (3) for type 4 entry
5
4
Input-output instruction:
(1) Space device to address field
(2) Print instruction operand contained in instruction word
3
2
Shift instruction:
(1) Space device to address field
(2) Print shift count contained in instruction word
1
Other machine instructions, no address output required
0
Enter PGEN
Print specified address in octal
If indexing indicator set print "I"; if not, print two spaces
Print a space
Search source address tables and INFORM address tables for specified address. If found, return corresponding name. (NFIN)
Decode and print name characters
Print three spaces
Print contents of specified address in octal
Exit PGEN

Figure 30. - Functional diagram of instruction list routine (LIST).

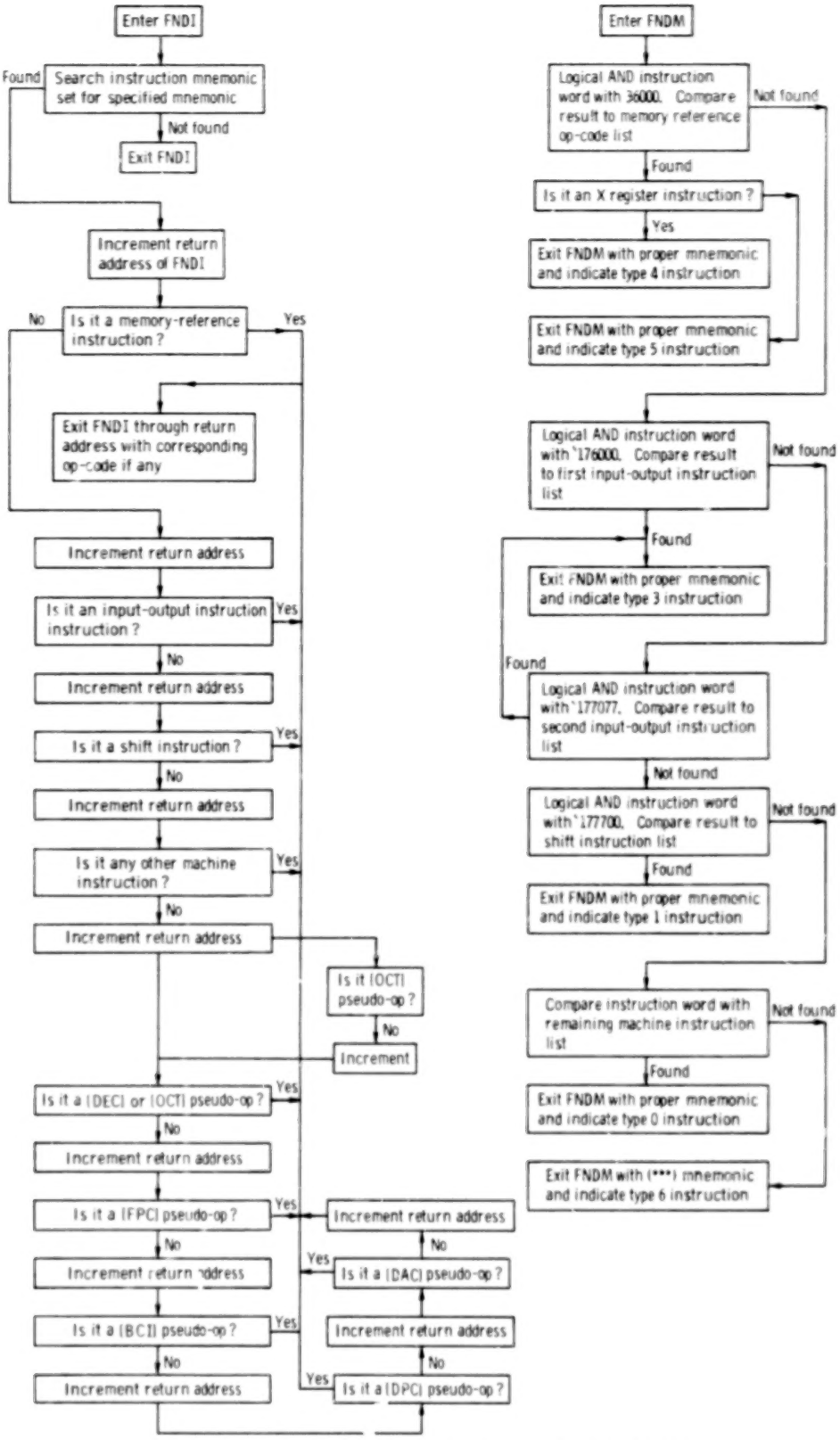

Enter FNDI
Found
Search instruction mnemonic set for specified mnemonic
Not found
Exit FNDI
Increment return address of FNDI
No
Is it a memory-reference instruction ?
Yes
Exit FNDI through return address with corresponding op-code if any
Increment return address
Is it an input-output instruction instruction ?
Yes
No
Increment return address
Is it a shift instruction ?
Yes
No
Increment return address
Is it any other machine instruction ?
Yes
No
Increment return address
Is it [OCT] pseudo-op ?
No
Increment
Is it a [DEC] or [OCT] pseudo-op ?
Yes
No
Increment return address
Is it a [FPC] pseudo-op ?
Yes
No
Increment return address
Is it a [BCI] pseudo-op ?
Yes
No
Increment return address
Increment return address
No
Is it a [DAC] pseudo-op ?
Yes
Increment return address
No
Is it a [DPC] pseudo-op ?
Yes
Enter FNDM
Logical AND instruction word with 36000. Compare result to memory reference op-code list
Not found
Found
Is it an X register instruction ?
Yes
Exit FNDM with proper mnemonic and indicate type 4 instruction
Exit FNDM with proper mnemonic and indicate type 5 instruction
Logical AND instruction word with `176000. Compare result to first input-output instruction list
Not found
Found
Exit FNDM with proper mnemonic and indicate type 3 instruction
Found
Logical AND instruction word with `177077. Compare result to second input-output instruction list
Not found
Logical AND instruction word with `177700. Compare result to shift instruction list
Not found
Found
Exit FNDM with proper mnemonic and indicate type 1 instruction
Compare instruction word with remaining machine instruction list
Not found
Found
Exit FNDM with proper mnemonic and indicate type 0 instruction
Exit FNDM with (***) mnemonic and indicate type 6 instruction

Figure 31. - Functional diagram of instruction search routines (FNDI) and (FNDM).

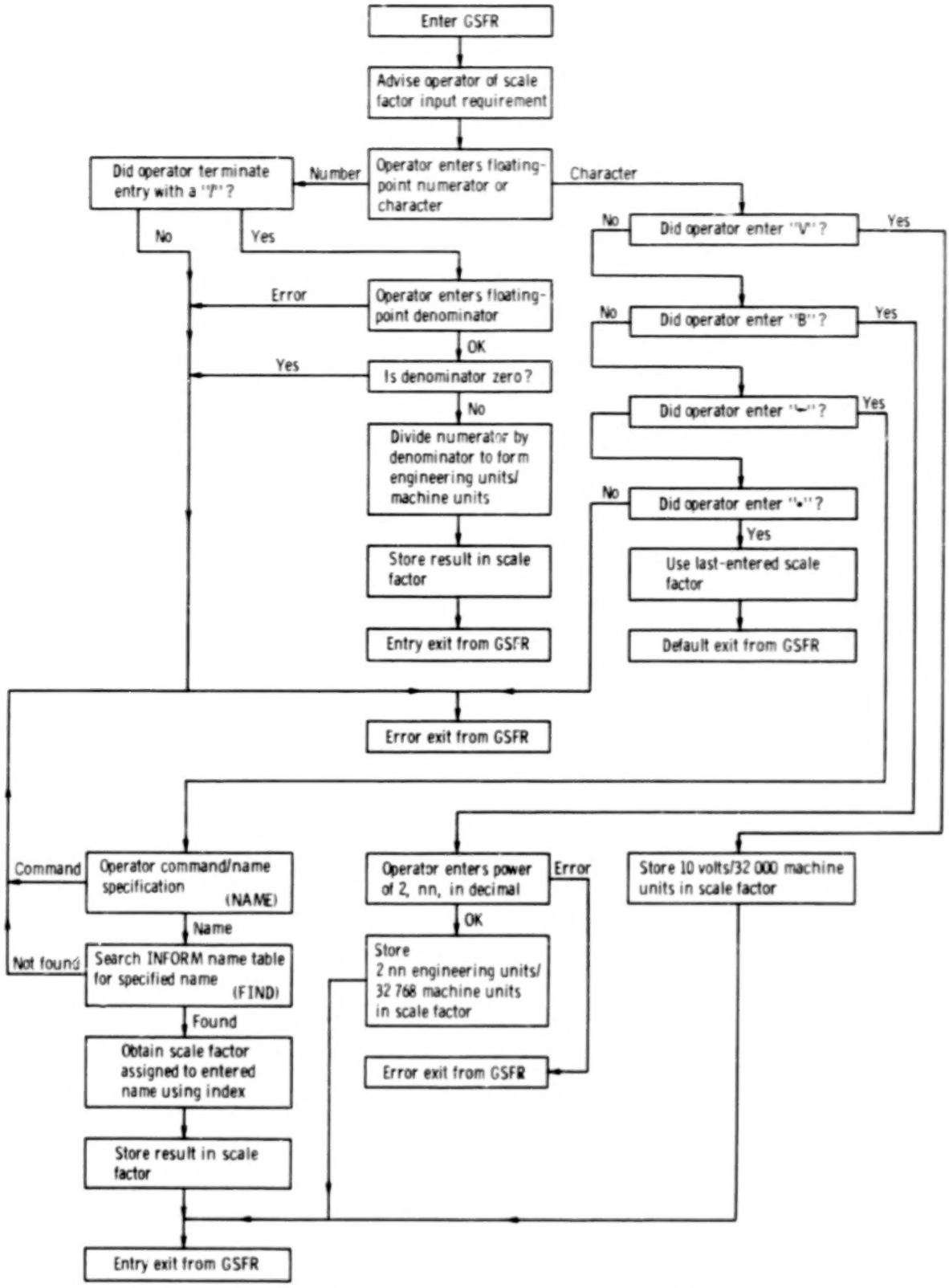


Figure 32. - Functional diagram of scale-factor input routine (GSFR).

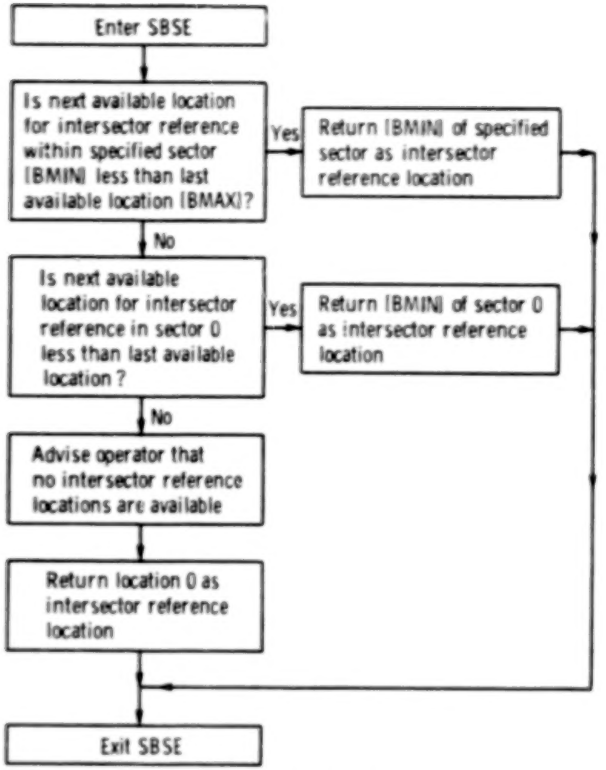

Enter SBSE
Is next available location for intersector reference within specified sector [BMIN] less than last available location [BMAX]?
Yes
Return [BMIN] of specified sector as intersector reference location
No
Is next available location for intersector reference in sector 0 less than last available location ?
Yes
Return [BMIN] of sector 0 as intersector reference location
No
Advise operator that no intersector reference locations are available
Return location 0 as intersector reference location
Exit SBSE

Figure 33. - Functional diagram of determination routine for intersector reference locations (SBSE).

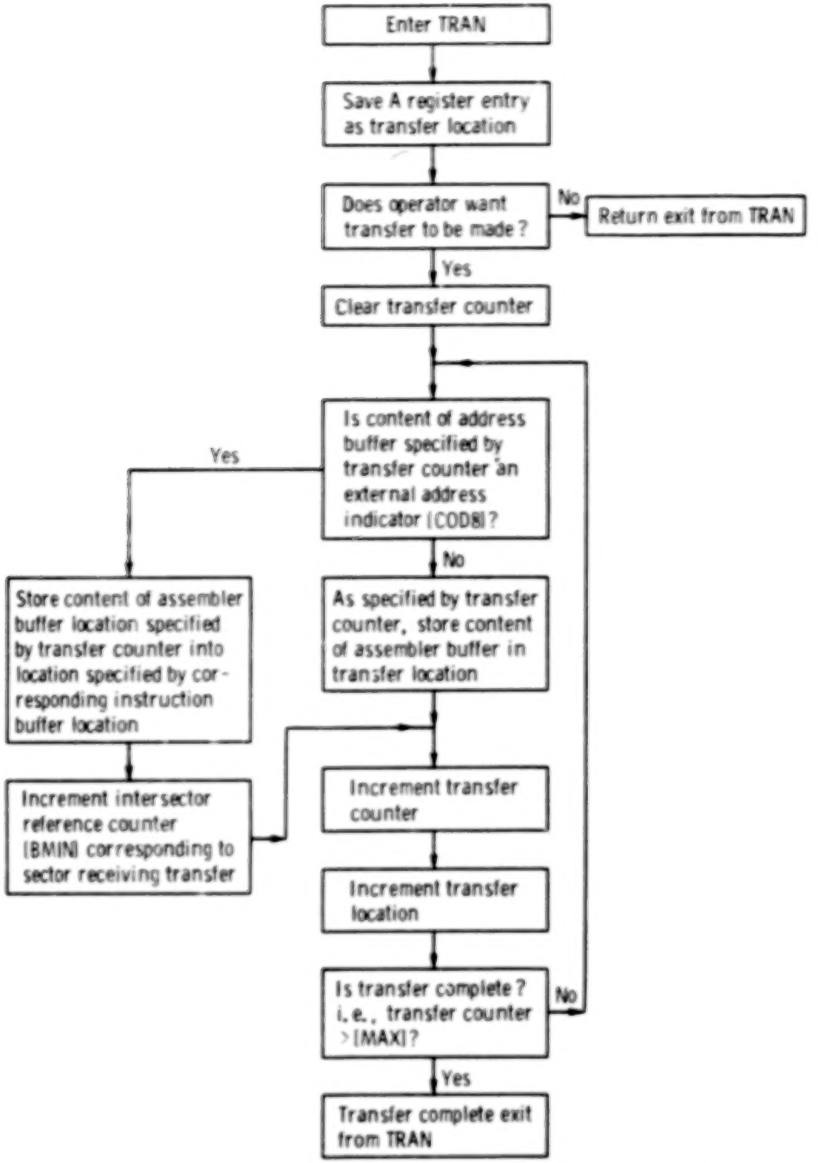

Enter TRAN
Save A register entry as transfer location
Does operator want transfer to be made ?
No
Return exit from TRAN
Yes
Clear transfer counter
Is content of address buffer specified by transfer counter an external address indicator (COD8)?
Yes
Store content of assembler buffer location specified by transfer counter into location specified by corresponding instruction buffer location
Increment intersector reference counter [BMIN] corresponding to sector receiving transfer
No
As specified by transfer counter, store content of assembler buffer in transfer location
Increment transfer counter
Increment transfer location
Is transfer complete ? i.e., transfer counter > [MAX]?
No
Yes
Transfer complete exit from TRAN

Figure 34. - Functional diagram of assembler buffer transfer routine (TRAN).

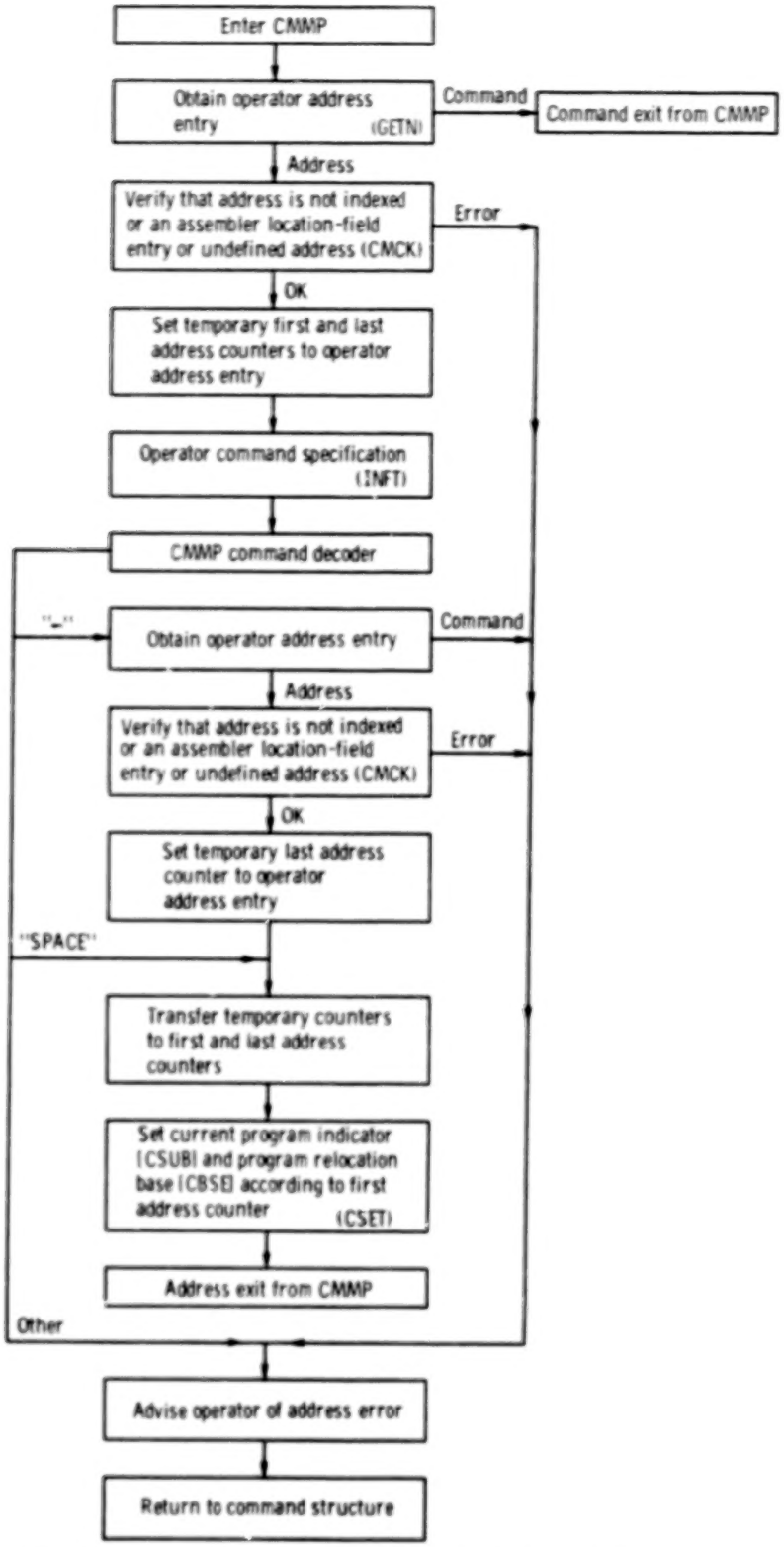

Enter CMMP
Obtain operator address entry (GETN)
Command
Command exit from CMMP
Address
Verify that address is not indexed or an assembler location-field entry or undefined address (CMCK)
Error
OK
Set temporary first and last address counters to operator address entry
Operator command specification (INFT)
CMMP command decoder
"-"
Obtain operator address entry
Command
Address
Verify that address is not indexed or an assembler location-field entry or undefined address (CMCK)
Error
OK
Set temporary last address counter to operator address entry
"SPACE"
Transfer temporary counters to first and last address counters
Set current program indicator (CSUB) and program relocation base (CBSE) according to first address counter (CSET)
Address exit from CMMP
Other
Advise operator of address error
Return to command structure

Figure 35. - Operator address-counter and command-specification routine (CMMP).

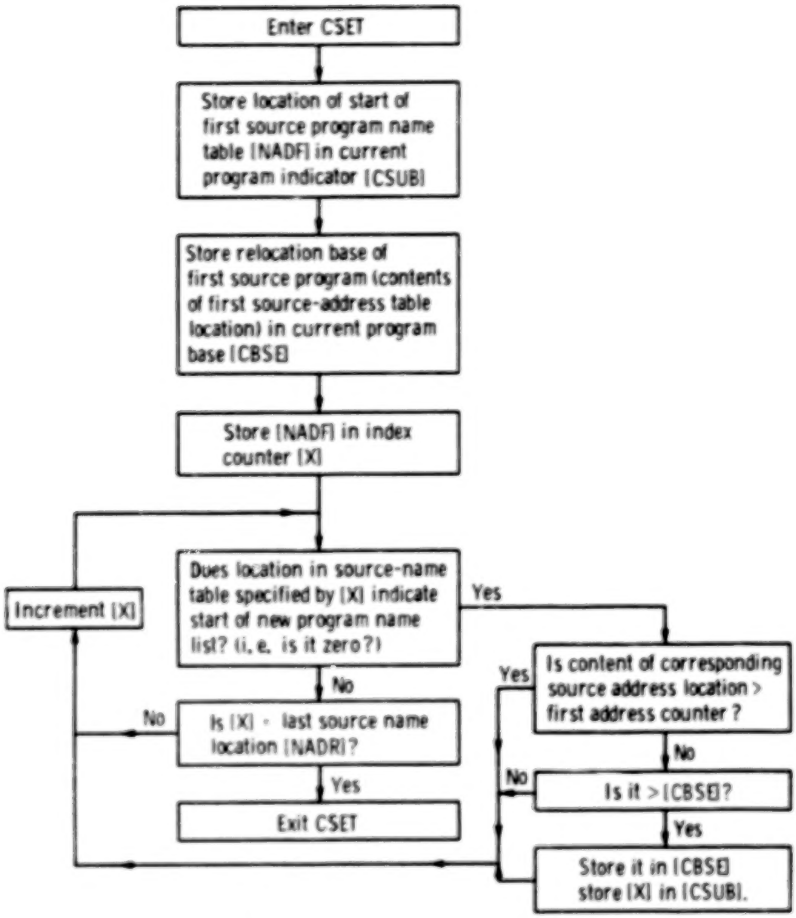

Enter CSET
Store location of start of first source program name table [NADF] in current program indicator [CSUB]
Store relocation base of first source program (contents of first source-address table location) in current program base [CBSE]
Store [NADF] in index counter [X]
Does location in source-name table specified by [X] indicate start of new program name list? (i. e. is it zero?)
Yes
No
Increment [X]
No
Is [X] = last source name location [NADR]?
Yes
Exit CSET
Is content of corresponding source address location > first address counter ?
Yes
No
Is it > [CBSE]?
No
Yes
Store it in [CBSE] store [X] in [CSUB].

Figure 36. - Functional diagram of current-program indicator and relocation-base initialization routine (CSET).

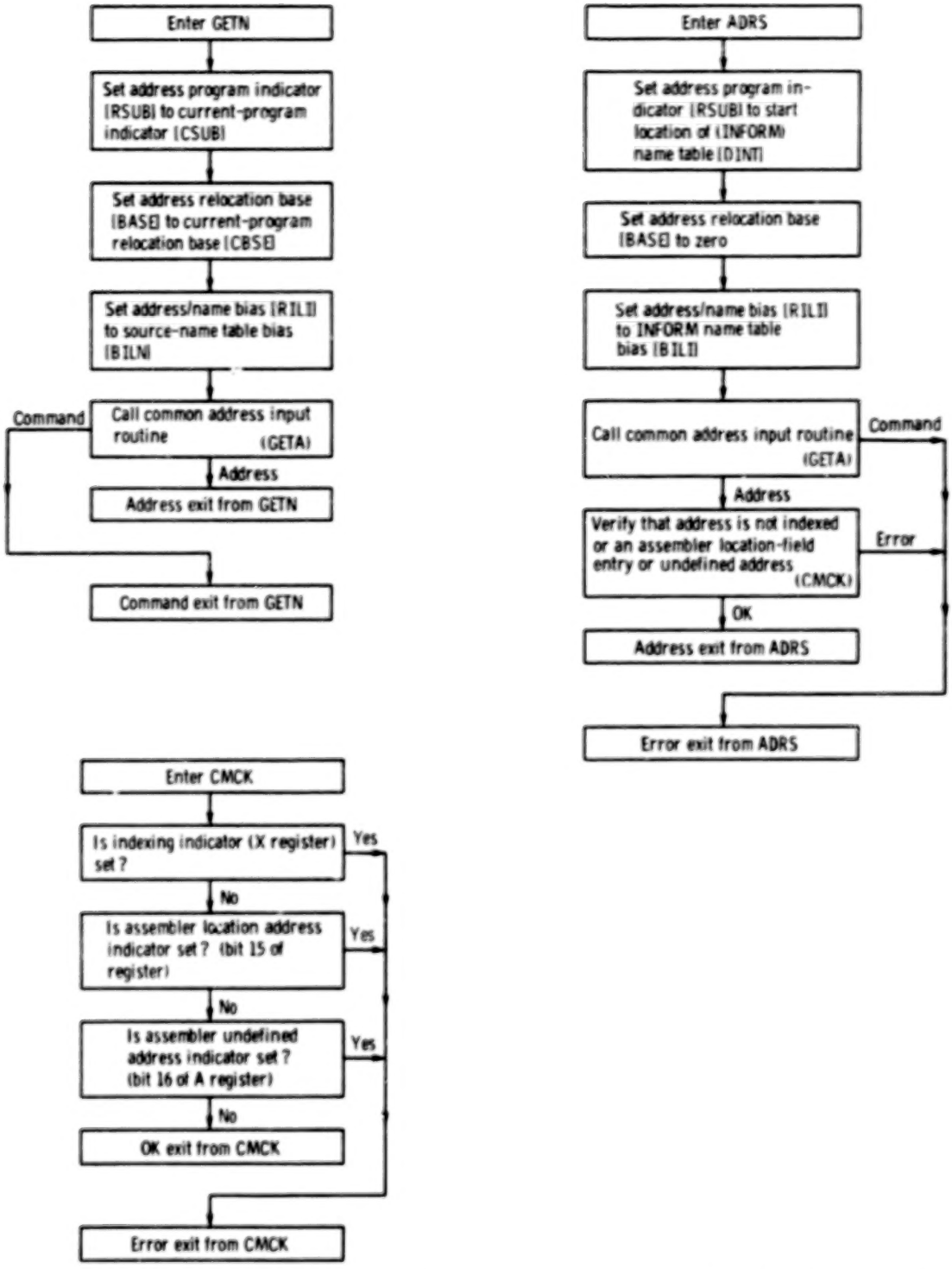

Enter GETN
Set address program indicator [RSUB] to current-program indicator [CSUB]
Set address relocation base [BASE] to current-program relocation base [CBSE]
Set address/name bias [RILI] to source-name table bias [BILN]
Command
Call common address input routine (GETA)
Address
Address exit from GETN
Command exit from GETN
Enter ADRS
Set address program indicator [RSUB] to start location of [INFORM] name table [DINT]
Set address relocation base [BASE] to zero
Set address/name bias [RILI] to INFORM name table bias [BILI]
Call common address input routine (GETA)
Command
Address
Verify that address is not indexed or an assembler location-field entry or undefined address (CMCK)
Error
OK
Address exit from ADRS
Error exit from ADRS
Enter CMCK
Is indexing indicator (X register) set ?
Yes
No
Is assembler location address indicator set ? (bit 15 of register)
Yes
No
Is assembler undefined address indicator set ? (bit 16 of A register)
Yes
No
OK exit from CMCK
Error exit from CMCK

Figure 37. - Functional diagram of operator address entry and verification routines (GETN), (ADRS), and (CMCK).

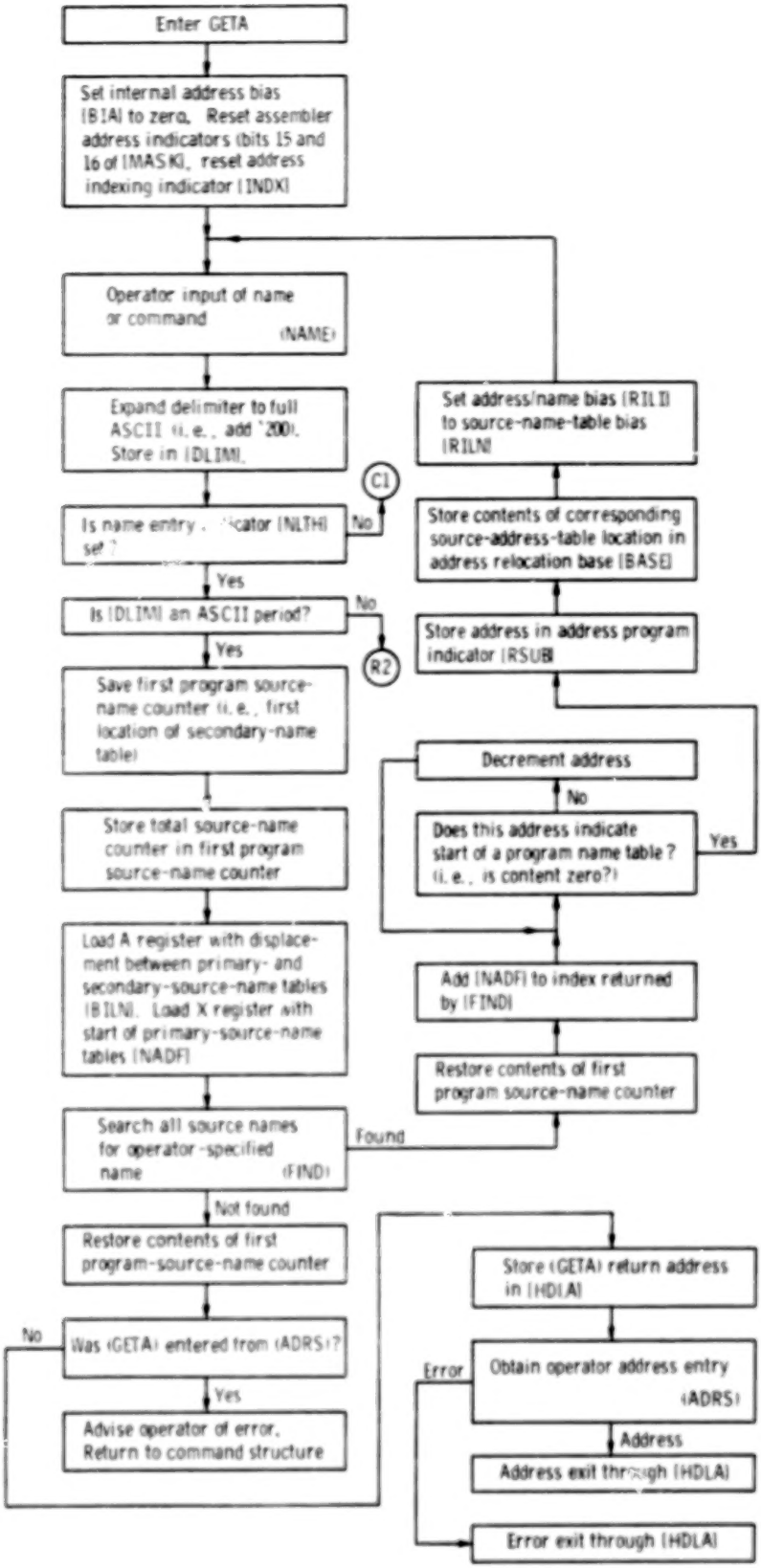

Enter GETA
Set internal address bias (BIA) to zero. Reset assembler address indicators (bits 15 and 16 of (MASK), reset address indexing indicator (INDX)
Operator input of name or command (NAME)
Expand delimiter to full ASCII (i.e., add '200). Store in (DLIM).
C1
Is name entry . 'icator (NLTH) set ?
No
Yes
Is (DLIM) an ASCII period?
No
Yes
R2
Save first program source-name counter (i.e., first location of secondary-name table)
Store total source-name counter in first program source-name counter
Load A register with displacement between primary- and secondary-source-name tables (BILN). Load X register with start of primary-source-name tables (NADF)
Search all source names for operator-specified name (FIND)
Found
Not found
Restore contents of first program-source-name counter
No
Was (GETA) entered from (ADRS)?
Yes
Advise operator of error. Return to command structure
Set address/name bias (RILI) to source-name-table bias (RILN)
Store contents of corresponding source-address-table location in address relocation base (BASE)
Store address in address program indicator (RSUB)
Decrement address
No
Does this address indicate start of a program name table? (i.e., is content zero?)
Yes
Add (NADF) to index returned by (FIND)
Restore contents of first program source-name counter
Store (GETA) return address in (HDLA)
Error
Obtain operator address entry (ADRS)
Address
Address exit through (HDLA)
Error exit through (HDLA)

Figure 38. - Functional diagram of common address input routine (GETA).

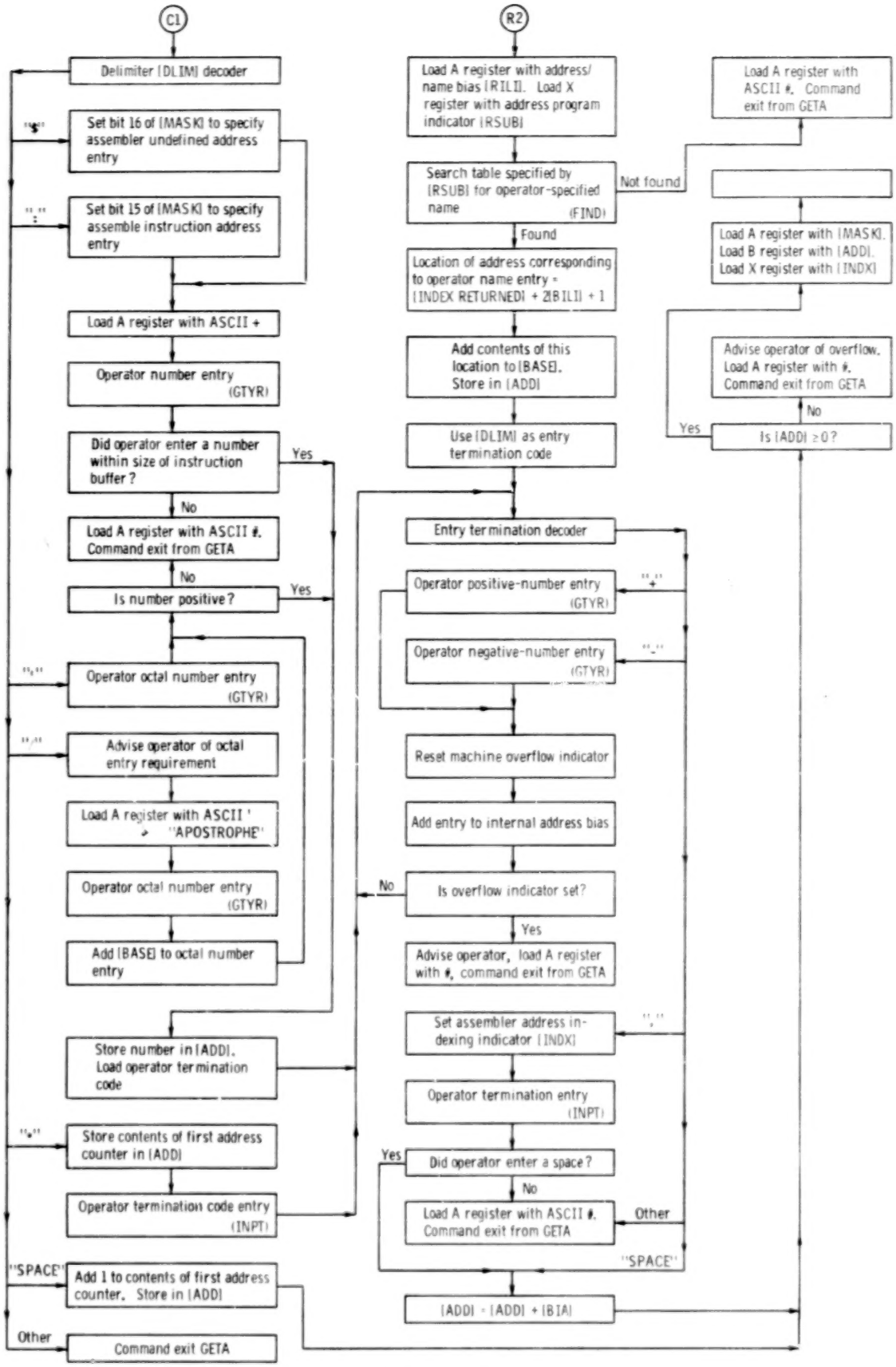

C1
Delimiter [DLIM] decoder
"$"
Set bit 16 of [MASK] to specify assembler undefined address entry
":"
Set bit 15 of [MASK] to specify assemble instruction address entry
Load A register with ASCII +
Operator number entry (GTYR)
Did operator enter a number within size of instruction buffer?
Yes
No
Load A register with ASCII #. Command exit from GETA
No
Is number positive?
Yes
"%"
Operator octal number entry (GTYR)
"/"
Advise operator of octal entry requirement
Load A register with ASCII ' = "APOSTROPHE"
Operator octal number entry (GTYR)
Add [BASE] to octal number entry
Store number in [ADD]. Load operator termination code
"."
Store contents of first address counter in [ADD]
Operator termination code entry (INPT)
"SPACE"
Add 1 to contents of first address counter. Store in [ADD]
Other
Command exit GETA
R2
Load A register with address/name bias [RILI]. Load X register with address program indicator [RSUB]
Search table specified by [RSUB] for operator-specified name (FIND)
Not found
Found
Location of address corresponding to operator name entry = [INDEX RETURNED] + 2[BILI] + 1
Add contents of this location to [BASE]. Store in [ADD]
Use [DLIM] as entry termination code
Load A register with ASCII #. Command exit from GETA
Load A register with [MASK]. Load B register with [ADDI. Load X register with [INDX]
Advise operator of overflow. Load A register with #. Command exit from GETA
No
Yes
Is [ADDI ≥ 0?
Entry termination decoder
"+"
Operator positive-number entry (GTYR)
"-"
Operator negative-number entry (GTYR)
Reset machine overflow indicator
Add entry to internal address bias
No
Is overflow indicator set?
Yes
Advise operator, load A register with #, command exit from GETA
","
Set assembler address indexing indicator [INDX]
Operator termination entry (INPT)
Yes
Did operator enter a space?
No
Load A register with ASCII #. Command exit from GETA
Other
"SPACE"
[ADD] = [ADD] + [BIA]

Figure 38. - Continued.

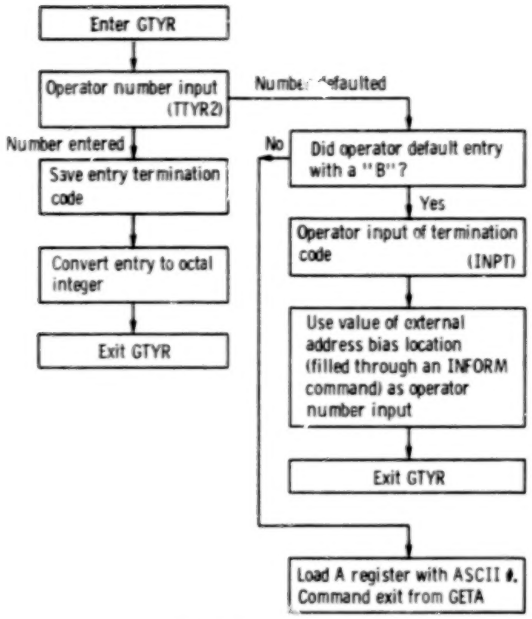

Enter GTYR
Operator number input (TTYR2)
Number defaulted
Number entered
Save entry termination code
Convert entry to octal integer
Exit GTYR
No
Did operator default entry with a "B"?
Yes
Operator input of termination code (INPT)
Use value of external address bias location (filled through an INFORM command) as operator number input
Exit GTYR
Load A register with ASCII #. Command exit from GETA

Figure 38. - Concluded.

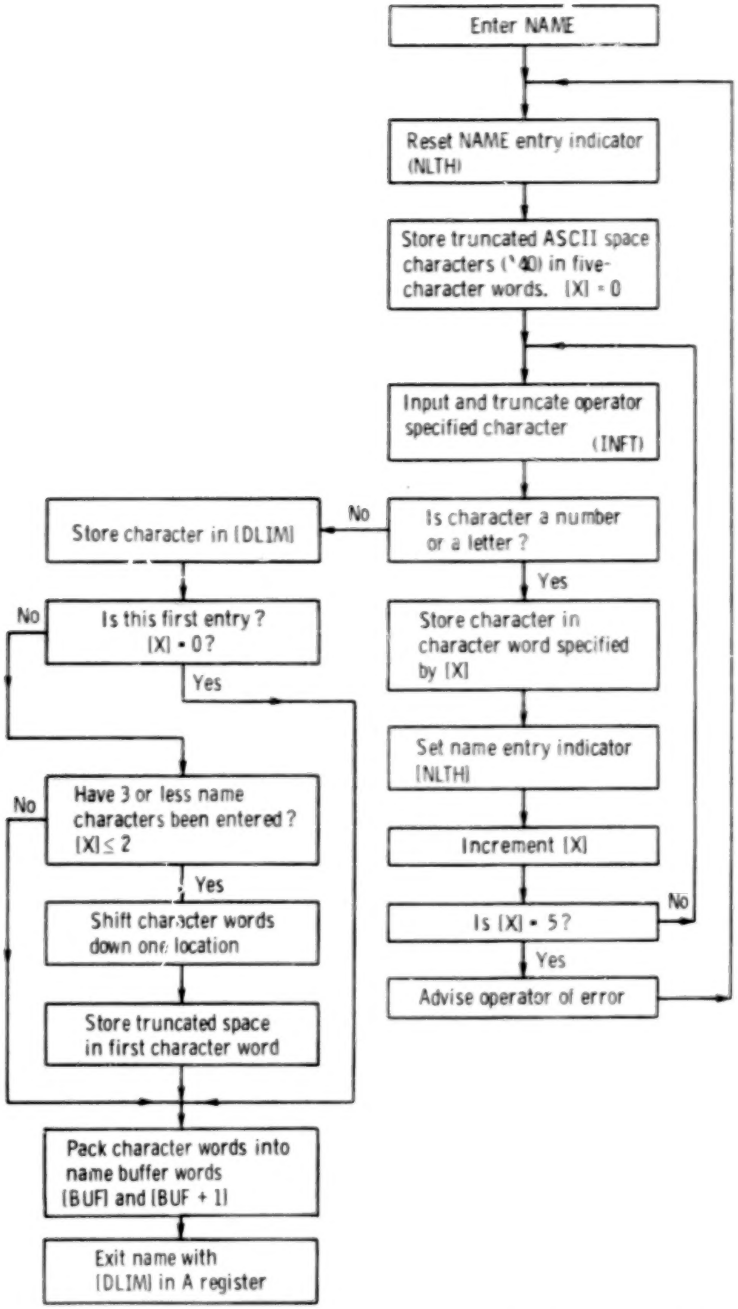

Enter NAME
Reset NAME entry indicator (NLTH)
Store truncated ASCII space characters ('40) in five-character words. [X] = 0
Input and truncate operator specified character (INFT)
Is character a number or a letter ?
No
Store character in [DLIM]
Yes
Store character in character word specified by [X]
Set name entry indicator [NLTH]
Increment [X]
Is [X] = 5 ?
No
Yes
Advise operator of error
Is this first entry ? [X] = 0 ?
No
Yes
Have 3 or less name characters been entered ? [X] ≤ 2
No
Yes
Shift character words down one location
Store truncated space in first character word
Pack character words into name buffer words [BUF] and [BUF + 1]
Exit name with [DLIM] in A register

Figure 39. - Functional diagram of name-command specification routine (NAME).

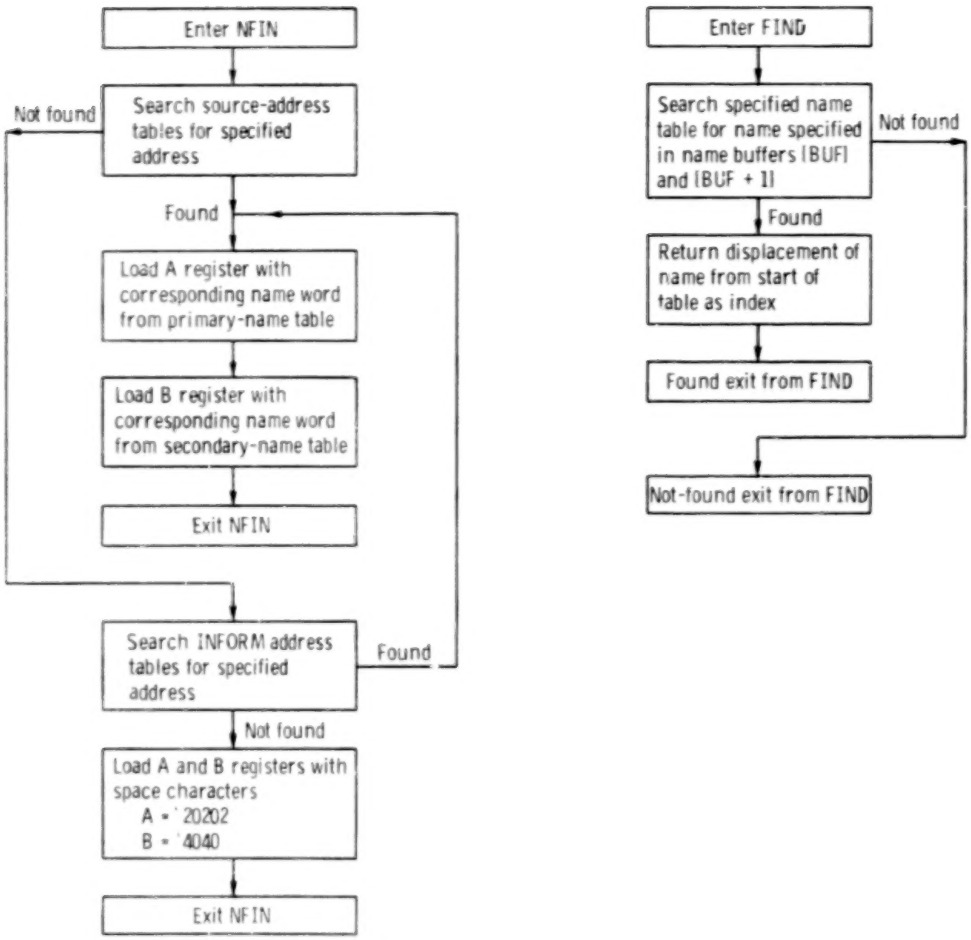

Enter NFIN
Search source-address tables for specified address
Not found
Found
Load A register with corresponding name word from primary-name table
Load B register with corresponding name word from secondary-name table
Exit NFIN
Search INFORM address tables for specified address
Found
Not found
Load A and B registers with space characters
A = '20202
B = '4040
Exit NFIN
Enter FIND
Search specified name table for name specified in name buffers [BUF] and [BUF + 1]
Not found
Found
Return displacement of name from start of table as index
Found exit from FIND
Not-found exit from FIND

Figure 40. - Functional diagram of name table search routines (NFIN) and FIND).

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TP-1441		

4. Title and Subtitle	5. Report Date
INTERACTIVE DEBUG PROGRAM FOR EVALUATION AND MODIFICATION OF ASSEMBLY-LANGUAGE SOFTWARE	April 1979
	6. Performing Organization Code
7. Author(s)	8. Performing Organization Report No.
Dale J. Arpasi	E-9219
	10. Work Unit No.
9. Performing Organization Name and Address	505-05
National Aeronautics and Space Administration Lewis Research Center Cleveland, Ohio 44135	11. Contract or Grant No.
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address	Technical Paper
National Aeronautics and Space Administration Washington, D.C. 20546	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

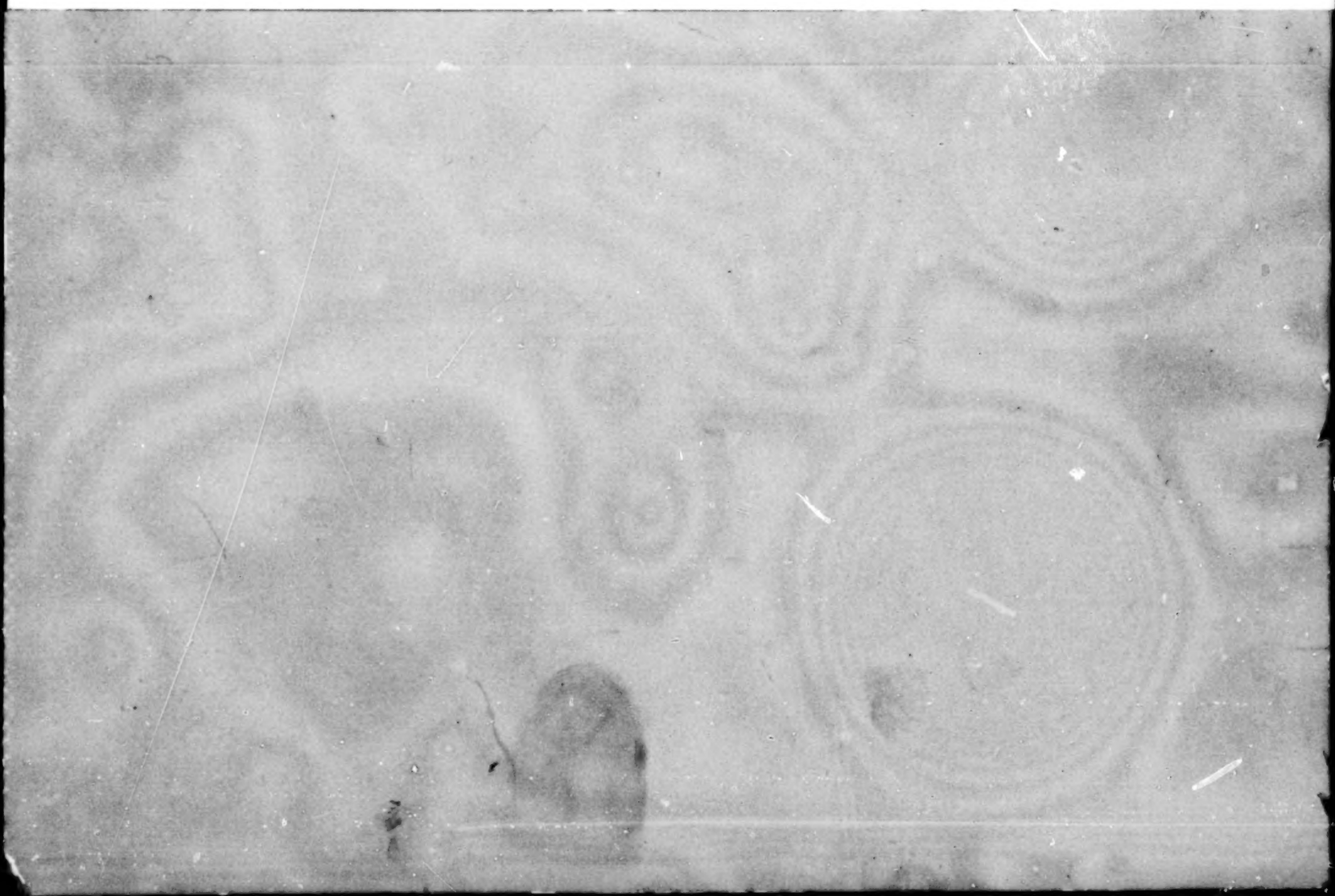
An assembly-language debug program written for the Honeywell HDC-601 and DDP-516/316 computers is described in detail. The program uses names and relative addressing to improve operator-machine interaction. Features include versatile display, on-line assembly, and improved program execution and analysis. The program is discussed from both a programmer's and an operator's standpoint. Functional diagrams are included to describe the program, and each command is illustrated.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Computer programs Honeywell computers Automatic control	Unclassified - unlimited STAR Category 61

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
Unclassified	Unclassified	80	A05

* For sale by the National Technical Information Service, Springfield, Virginia 22161

NASA-Langley, 1979



END

June 18, 1981